Sistemas Informáticos
Curso 2004-2005

Modelación de un sistema no lineal de vuelo

Mario Chueca Burgueño
Javier López Carreras
Iván Rebollo Martínez

Dirigido por:
Prof. Jesús Manuel de la Cruz García
Dpto. de Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

ÍNDICE

1 RESUMEN

1.1 ESPAÑOL

El proyecto consiste en un simulador de helicóptero desarrollado en los entornos de programación Matlab y Simulink. Un simulador con opciones para controlar con el teclado, con un joystick virtual o con un joystick analógico. Asimismo cuenta con control de colisiones para dar un mayor realismo al proyecto, mundo infinito para no tener restricciones de terreno y varias posibles visualizaciones de los resultados como pueden ser los diales... Tiene capacidad para control manual y para varios tipos de pilotos automáticos con la posibilidad de ir cambiando de uno a otro de forma inmediata.

También cuenta con diferentes escenarios, los cuales se pueden intercambiar, para hacer más entretenida la parte visual de la aplicación y dentro de los cuales se encuentran los diferentes objetivos a donde llegar con el helicóptero, haciendo que el simulador sea realmente entretenido.

1.2 ENGLISH

The project consists of an helicopter simulator developed in Matlab and Simulink. A simulator with options to control with the keyboard, virtual joystick or analogical joystick. Also counts with control of collisions to give a greater realism to the project, infinite world to do not have land restrictions and several possible visualizations of the results as can be the dials... It has capacity for manual control and for several types of automatic pilots with the possibility of change to one another in live-time execution.

It counts with different scenes, which can be interchanged, that makes the visual part more entertaining. In those scenes we can find the different objectives where we have to arrive with the helicopter, causing the simulator to be really entertaining.

2 INTRODUCCIÓN

Este proyecto trata de la simulación de un modelo no lineal de vuelo, realizado en el entorno Matlab y Simulink. Como punto de partida, teníamos dicho modelo de un proyecto anterior del Departamento de Arquitectura de Computadores y Automática. Hemos llevado un desarrollo con algunos momentos de colaboración con el creador del modelo. Que nos permitió ver su posible uso real, puesto que este departamento esta trabajando en la creación de una maqueta real de helicóptero. En el siguiente trabajo se va a intentar desarrollar las diferentes características que tienen los actuales simuladores, y en particular el nuestro.

En primer lugar se va a desarrollar el tema de la **evolución** que se ha llevado en la realización del proyecto, en donde se podrá entender con mayor detalle el proceso de creación que ha sufrido este proyecto.

Posteriormente se habla del **modelo** matemático, así como unas notaciones de la física del vuelo del helicóptero, mediante el modelo de simulink del que partíamos.

También cabe resaltar el **manual de usuario** para poder instalar y ejecutar correctamente el simulador, así como aprender a controlarlo mediante las teclas propuestas. Esta parte será muy útil, por si alguien se decide a continuar nuestro proyecto, y será ahí donde encontrará lo básico para poder moverse con soltura.

A continuación viene el grueso de la **implementación** del proyecto, en el cual explicamos las propiedades y características de cada una de las partes implicadas, básicamente dos, el mundo virtual y el modelo de Simulink con los pertinentes detalles de cada una de ellas.

Para finalizar comentamos las **conclusiones** del proyecto así como unas posibles mejoras que se pueden realizar.

3 LA EVOLUCIÓN

El desarrollo temporal de nuestro proyecto se podría separar en varias etapas. Lo primero que hay que saber es que de nosotros, solo uno había tocado Matlab con anterioridad, y en ningún caso se tenía idea de las posibilidades que ofrecía.

También tuvimos algunos problemas como el no tener las toolboxes necesarias en los laboratorios de la facultad, con lo que hubo que repartir trabajos, y plantear reuniones cada cierto tiempo para juntarlas.

Por esta razón el comienzo fue lento, cada uno de nosotros se fue mirando por su lado, intentando absorver lo más posible de las ayudas de Matlab y de los propios ejemplos del programa. Además de este conocimiento que debíamos adquirir, nos hacía falta entender el modelo con el que partíamos, y hacernos con el lenguaje y con las nociones básicas necesarias sobre helicópteros, de los que tampoco conocíamos más de lo visto en documentales y películas.

En ese momento hicimos nuestro primer mundo virtual, muy básico, además de intentar con unos sliders controlar al helicóptero. Aquí empezaron los peores meses de nuestro proyecto, fue la fase en la que nos centramos en intentar controlar el helicóptero, lo cual se nos hacía imposible, las semanas pasaban sin avances, llegando a pensar que el modelo estaba mal, lo cual nos hubiera dejado sin opciones de realizar un simulador real, y nos hubiéramos debido quedar en un juego. En esta etapa aprovechamos para profundizar en la creación de mundos virtuales.

Había pasado medio año, cuando quedamos con Guillermo Martínez, que fue el creador del modelo, para comentar nuestras dudas sobre el mismo. Tras diferentes pruebas, nos confirmó la inestabilidad del helicóptero, y fue ahí cuando empezaron a surgir todo tipo de ideas, como los controles automáticos.

Llegados a este punto, ya dejamos de preocuparnos tanto por el control manual, que parecía lo más básico, y que acaparaba toda nuestra atención, y empezamos a trabajar en todo lo demás, tipos de entradas(teclado y joysticks), colisiones, mundo infinito, diferentes tipos de cámaras... Cosas que en principio no tenían mucho sentido, si al helicóptero no se le podía ni mover.

Aquí cometimos el error de no tener versiones sencillas del proyecto de manera que algunas pruebas eran bastante costosas. Esto se debe a que como hemos dicho antes Matlab ofrece muchas posibilidades, pero también según iba avanzando el proyecto iba consumiendo más recursos, hasta el punto que algunas versiones sólo dejaba probarlas una vez sin bloquear el ordenador.

Desde este momento hasta el final, fue la parte más ilusionante y constructiva de nuestro proyecto, llegando al final con el resultado obtenido, del que nos sentimos bastante orgullosos, con la única pega de las cosas que no dieron tiempo, de las que hablaremos más en la parte de conclusiones y posibles mejoras.

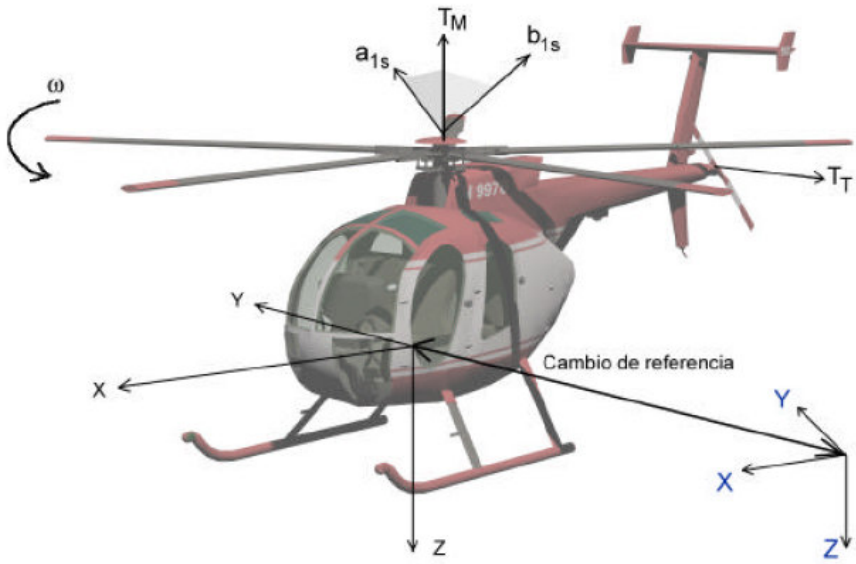
4 EL MODELO

El modelo que utilizamos nos permite modelar el sistema para controlar el vuelo de una maqueta de helicóptero. El hecho de que sea una maqueta hace que en el modelo de simulink pongamos un peso de 5 kilogramos.

En un principio vamos a hablar de las variables de entrada al sistema, que sirven para poder controlar el helicóptero:

- Tm: Empuje del rotor principal, es decir, la fuerza que mueve las hélices principales. La magnitud del vector empuje del rotor principal.
- TT: Empuje del rotor de cola, es decir, la fuerza que mueve las hélices traseras. La magnitud del vector empuje del rotor de cola.
- A_{1s}: Inclinación longitudinal del vector de empuje de las aspas del rotor principal.
- B_{1s}: Inclinación lateral del vector de empuje de las aspas del rotor principal.

Se puede ver que las fuerzas que hacen al helicóptero moverse son aquellas originadas por el empuje ejercido por ambos rotores, condicionadas por la inclinación de las aspas.

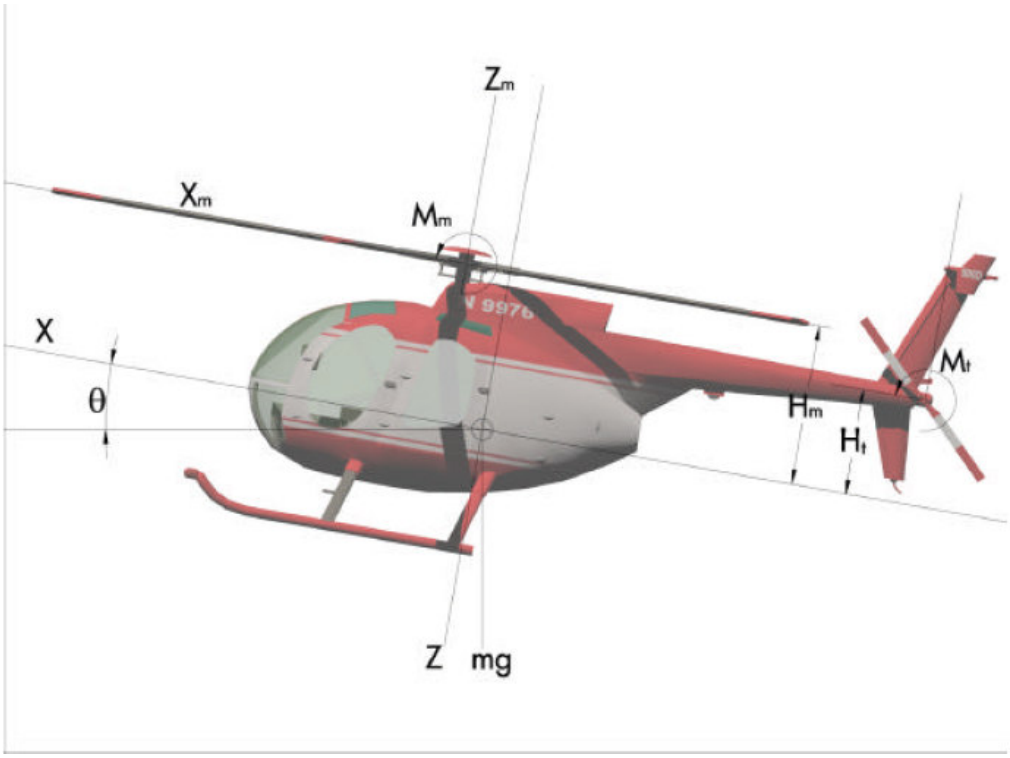


Y con estas variables de entrada, el modelo produciría las siguientes variables de salida:

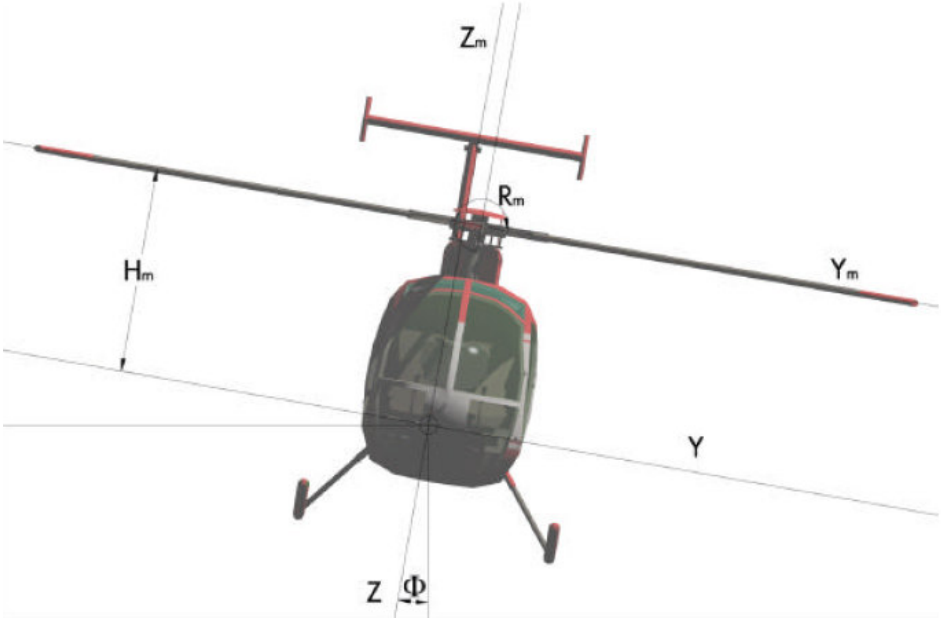
- P - Vector de posiciones referido en coordenadas espaciales Norte-Este-Abajo.
- v^p - Vector de velocidades referido en coordenadas espaciales Norte-Este-Abajo.
- R - Vector de rotación de la aeronave referido en ángulos de Euler, parametrizados en ZYX.
- w^b - Vector de velocidades angulares, referido en ejes cuerpo.

Aparte de la posición del helicóptero, el modelo devuelve tres ángulos para determinar la rotación del helicóptero. Estos ángulos son: pitch, roll y yaw.

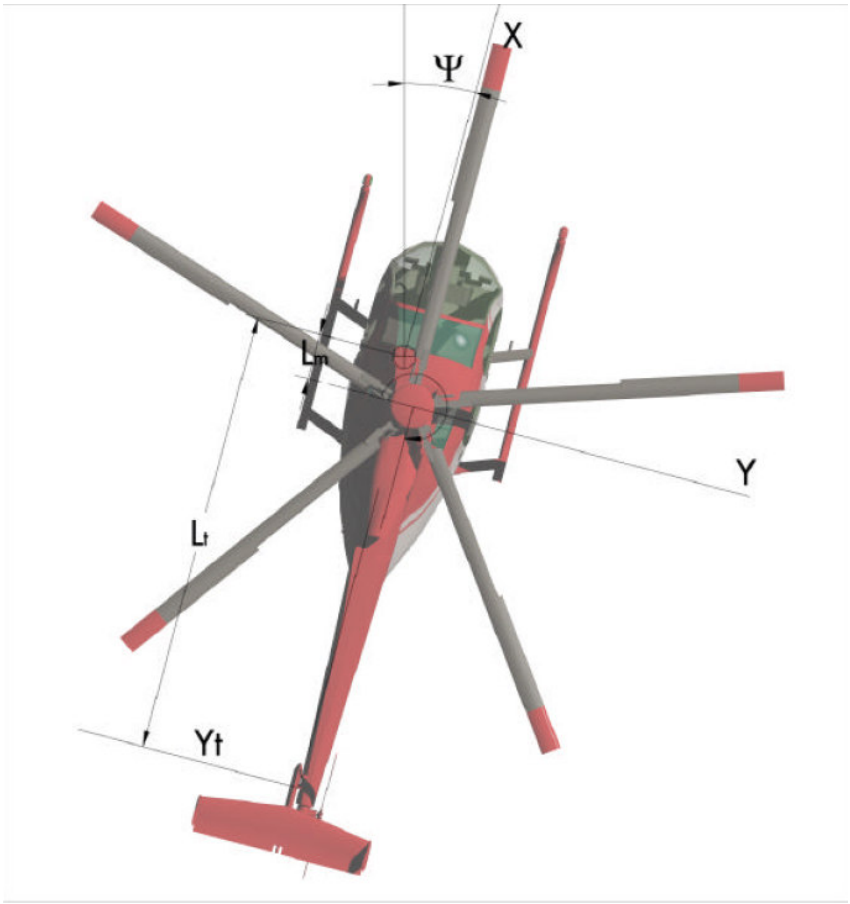
El pitch o θ, es el ángulo de rotación a través del eje y:



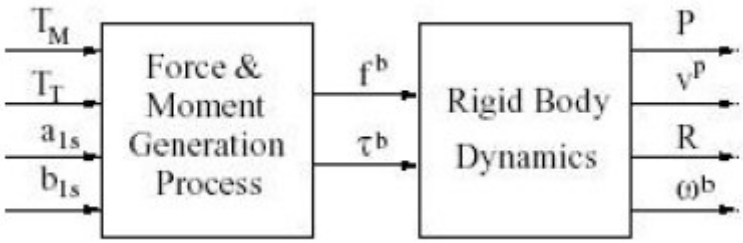
El roll o Φ, es el ángulo de rotación a través del eje x:



El yaw o Ψ, es el ángulo de rotación a través del eje z:



Como breve resumen del modelo matemático que utilizamos(basado en el modelo matemático utilizado en MODELACIÓN DEL SISTEMA NO LINEAL DE UN HELICÓPTERO – proyecto del departamento de arquitectura de computadores y automática de la UCM) se puede decir que se divide en dos subsistemas, generación de fuerzas y momentos, y dinámica del cuerpo rígido:



Cabe destacar las variables intermedias f^b y T^b:

- f^b: Vector de fuerzas totales ejercidas sobre el fuselaje de la aeronave referidas en *ejes cuerpo.*
- T^b: Vector de momentos angulares de rotación totales ejercidos sobre el fuselaje referidos en *ejes cuerpo.*

El sistema de referencia de la maqueta, se encuentra en el centro de gravedad de la misma, el eje x esta apuntando a la proa del helicóptero, el eje y apunta hacia la izquierda desde el centro de gravedad y el eje x apunta hacia abajo desde este punto.

A partir de estas variables y con este sistema de referencias, se le aplican las ecuaciones del movimiento para un cuerpo rígido sujeto a una fuerza y un momento angular.

La posición y velocidad del centro de gravedad del helicóptero están dadas por el vector posición $P \in \Re^3$ y el vector velocidad $v^p=\dot{P} \in \Re^3$, expresados en el sistema de referencia espacial orientado Norte-Este-Abajo.

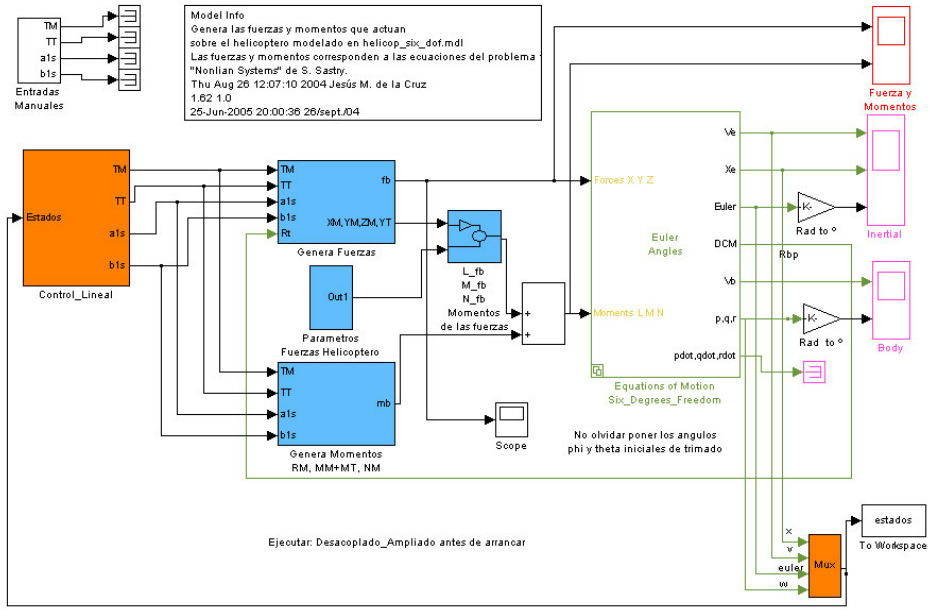
En cuanto a la generación de fuerzas y momentos, como hemos dicho antes, la fuerza que impulsa el movimiento del helicóptero viene dada por los rotores principal y de cola. Pero la generación de momentos lineales y angulares sobre el fuselaje puede ser considerada como la acción conjunta de varios elementos: el propio fuselaje, un estabilizador horizontal, un estabilizador vertical y el rotor de cola.

Una vez tenemos el modelo, utilizamos unos valores de trimado del helicóptero, que nos fueron dados por el propio proyecto.

Los valores de trimado del helicóptero son aquellos en los que los momentos angulares de rotación y las fuerzas que actúan sobre este son ambas cero. Para estos valores, el helicóptero permanece volando sobre el mismo punto, y no varían su posición, ni su aceleración ni su orientación.

Los valores utilizados en el modelo, están en las variables TM_trim, TT_trim, a1s_trim, b1s_trim. Y son 48.953, 2.4674, -0.01815 y 0.0061967 respectivamente.

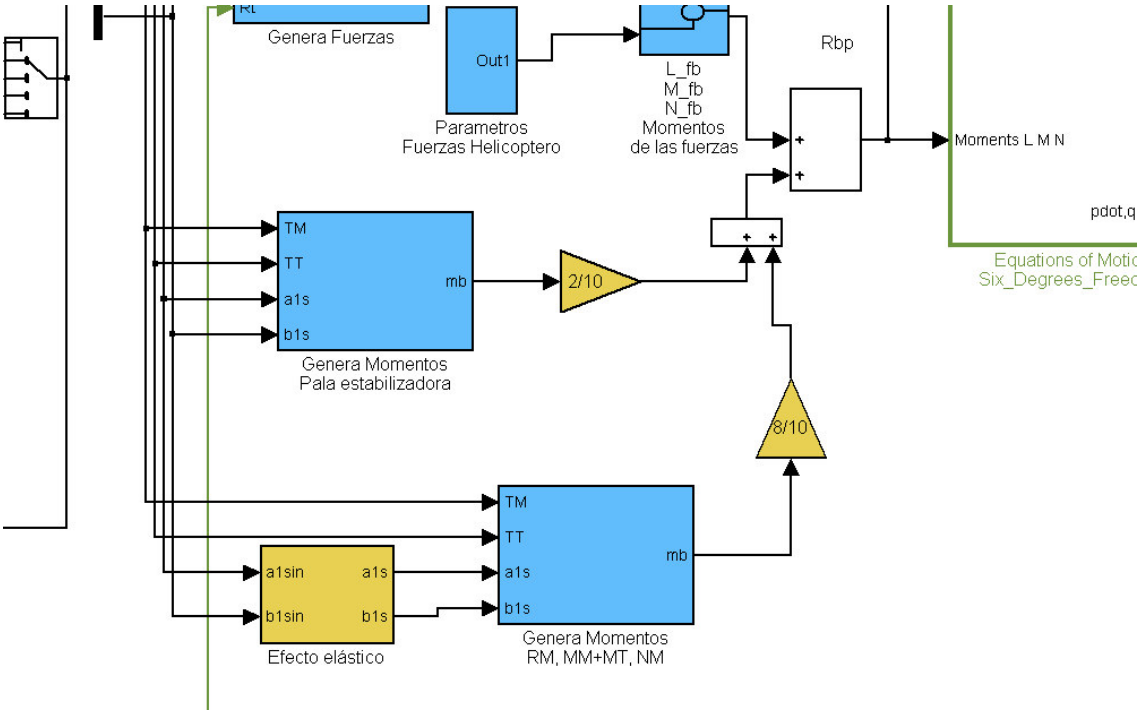
4.1 SIMULINK



En un principio utilizamos un modelo de simulink ya hecho a partir del cual se podría controlar el vuelo de una maqueta de helicóptero.

A este modelo se le añadió una pala estabilizadora que hiciera que los movimientos no fueran tan bruscos y que originase una fuerza inversa al movimiento requerido, para intentar estabilizar el helicóptero.

Ésta fuerza estabilizadora no debe de ser tan grande como la que genera el movimiento, por tanto se utiliza un amortiguamiento que se puede regular según se prefiera. El amortiguamiento del movimiento y la estabilizadora, ha de sumar uno, es decir, la fuerza del movimiento pueden ser 8/10 y el amortiguamiento de la pala estabilizadora 2/10.



A partir de este modelo, lo fuimos mejorando para poder controlar y generar rumbos de manera que nos quedó el modelo de simulink que se muestra con mayor detalle más adelante.

5 MANUAL DE USUARIO

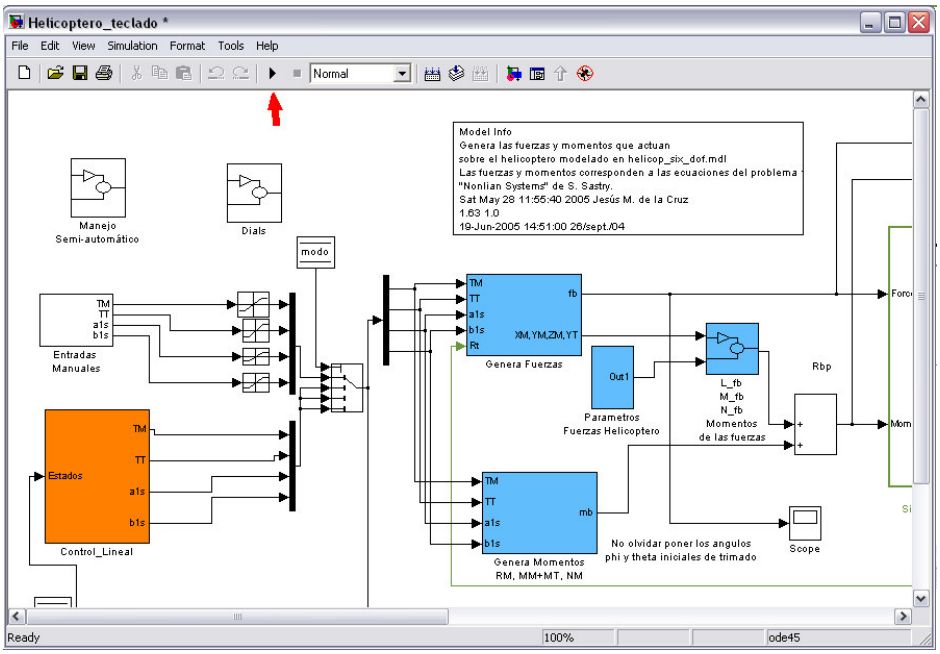
5.1 INSTALACIÓN

Para la instalación correcta de la aplicación se necesita instalar previamente **MATLAB 6.5** (es importante que sea la versión 6.5 o superior pero que tenga Virtual Reality toolbox, ya que esta es la necesaria para la correcta ejecución del producto).

Todos los archivos de la aplicación se deben de instalar en **c:/Proyecto** ya que sino se producirán errores a la hora de arrancar el programa.

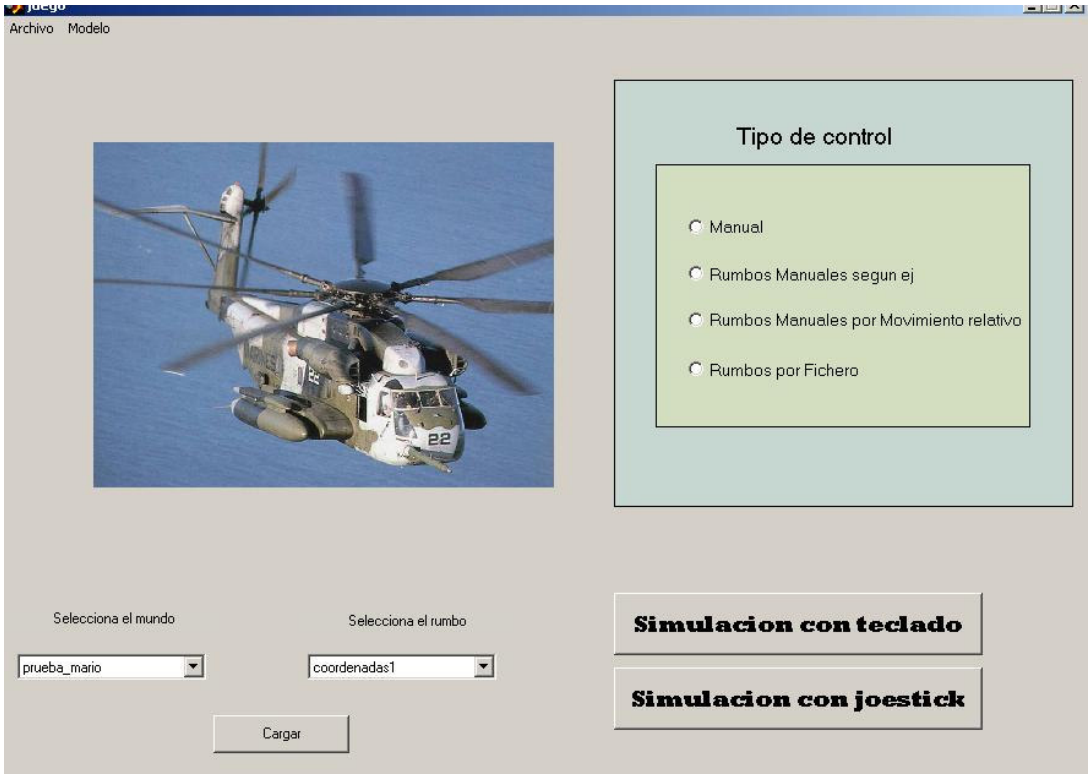
5.2 EJECUCIÓN

Para ejecutar el programa se tendrá que acceder al directorio donde se encuentran los archivos (**c:/Proyecto**) y desde allí hacer doble clic en el archivo **Helicóptero_teclado.** Con esto se abrirá el programa MATLAB y se abrirá el modelo en SIMULINK. Ya solo nos faltara darle al play en la ventana de SIMULINK.(Ver donde indica la flecha).



Ejecución a través de la GUI

También se puede ejecutar el programa a través de la GUI, arrancando MATLAB introduciendo en la consola la palabra juego. Con esto se nos abrirá la GUI en la que podemos elegir el escenario a cargar, el archivo desde donde se cogen las coordenadas de los objetivos y el modo de control que queramos tener sobre el helicóptero. Para que los cambios producidos en escenarios y coordenadas tengan efecto hay que pulsar el botón cargar. Cuando hayamos seleccionado estas tres propiedades tendremos que dar a uno de los botones para comenzar la simulación, bien con joestick o con teclado, con lo que se abrirá MATLAB y SIMULINK y ya solo faltará darle al botón del play como se ha visto anteriormente.



5.3 CONTROLES

El control del helicóptero se puede hacer de distintas formas:

-Por teclado:

- 1,2,3,4: Pulsando estas teclas cambiamos el modo del helicóptero.

  1:Manual

  2:Manual según ejes

  3:Manual según orientación

  4:Automático

- 7,8,9: Pulsando estas teclas cambiamos las vistas del helicóptero

  7:Vista trasera

  8:Vista Objetivo

  9:Vista Circular o móvil

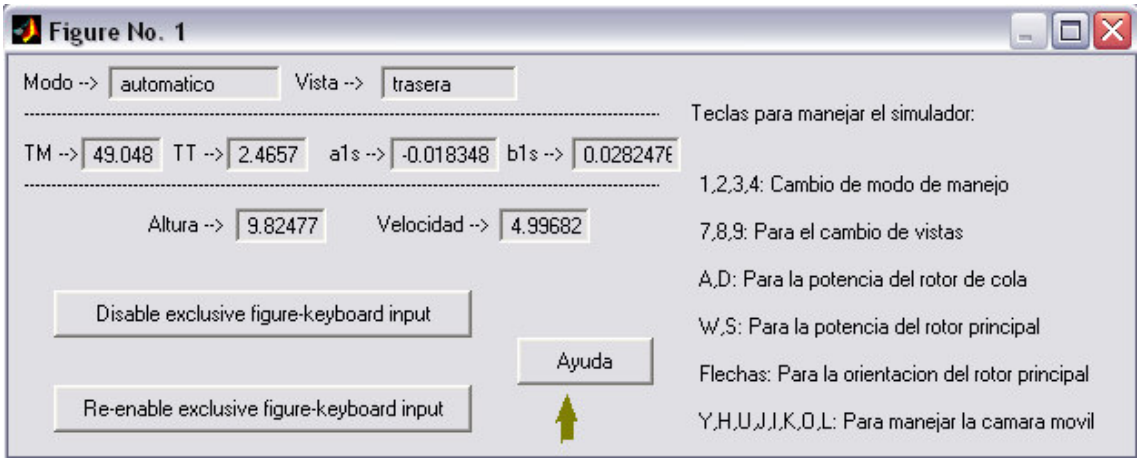
- A,D: Aumenta/disminuye la potencia del rotor de cola

- W,S: Aumenta/disminuye la potencia del rotor principal

- Cursores: Orientación del rotor principal

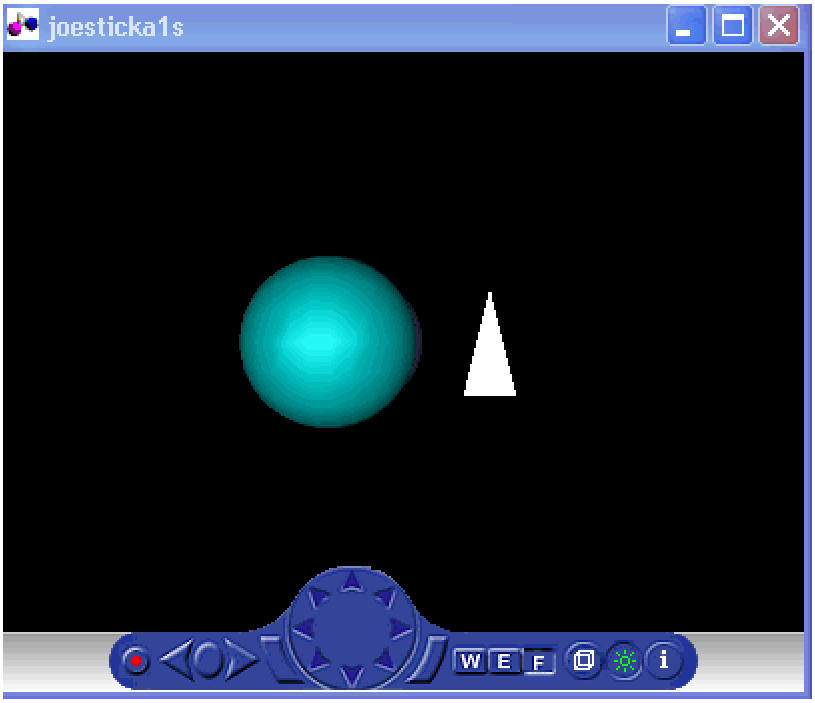
- Y,H,U,J,I,K,O,L: Manejo de la cámara circular o móvil

Si se hace clic en el botón de ayuda se despliega un panel con la ayuda de las funciones que realiza cada tecla.



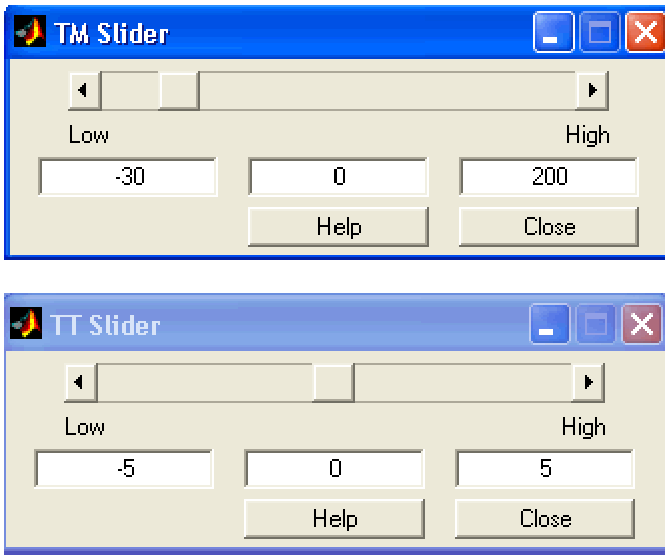
-Por Joestick virtual:

Para poder usar esta opción tendremos que abrir otro archivo. Hacer doble clic en **Helicop_six_dof_ctr_lineal** y así se abrirá MATLAB y SIMULINK. En la pantalla de MATLAB se tendrá que ejecutar el archivo inicio.m. Se escribirá en la consola de MATLAB la palabra inicio. El siguiente paso es darle al play de SIMULINK y se abrirá el joystick virtual dispuesto para controlar el helicóptero. También se puede abrir desde la GUI pulsando el botón Simulación con joestick.



Si movemos el joestick a izquierda/derecha controlamos b1s. Si lo movemos de arriba/abajo controlamos a1s.

Para controlar la TT y la TM necesitaremos usar dos sliders. Para abrirlos tendremos que ir al modelo en SIMULINK y hacer clic en el bloque Manejo Semi-automático y luego en los bloque TT Slider y TM Slider. Con esto abriremos los dos siguientes sliders y ya solo tendremos que mover el cuadrado con el ratón para seleccionar el valor que queramos.



-Por Joestick analógico:

Su uso dependerá de las características propias del joestick usado por el usuario.

5.4 PANTALLA

La pantalla del juego se divide en diferentes partes:

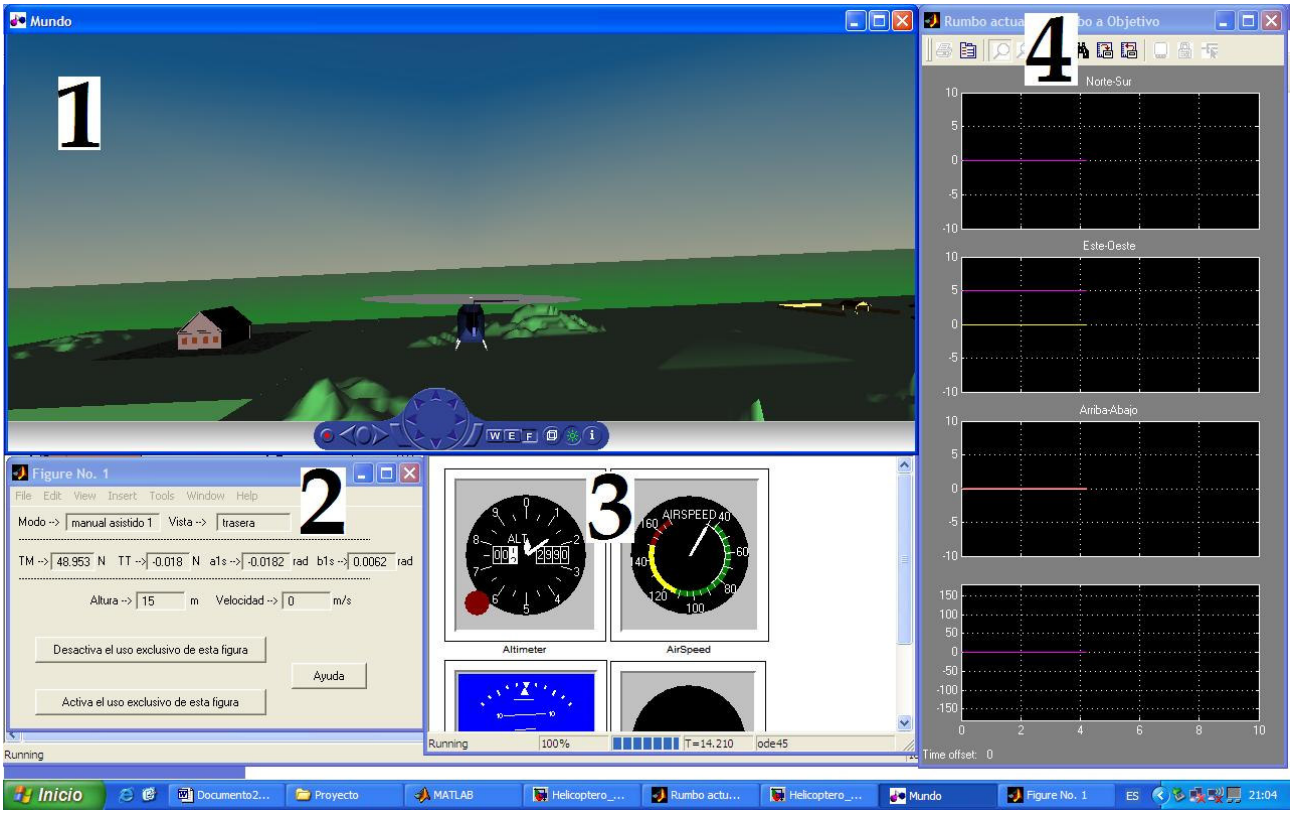
1.Visor de Realidad Virtual: donde se ve el escenario, el helicóptero y sus movimientos.

2.Cuadro de mandos: donde se encuentran el modo y la vista que se están usando la ayuda para los controles y los valores de velocidad, altura, a1,b1,tt y tm.

3.Diales: observamos los diales propios de un helicóptero como son un altímetro (nos indica la altura del helicóptero respecto al suelo)y el horizonte artificial(nos indica la posición del helicóptero respecto de sus ejes).

4.Scopes: Se pueden ver 4 Scopes. El primero nos indica el eje norte-sur. El segundo nos indica el eje este. El tercero será el eje arriba-abajo y el cuarto el de la inclinación del helicóptero.

Ver los números en la siguiente figura.



5.5 OBJETIVO DEL JUEGO

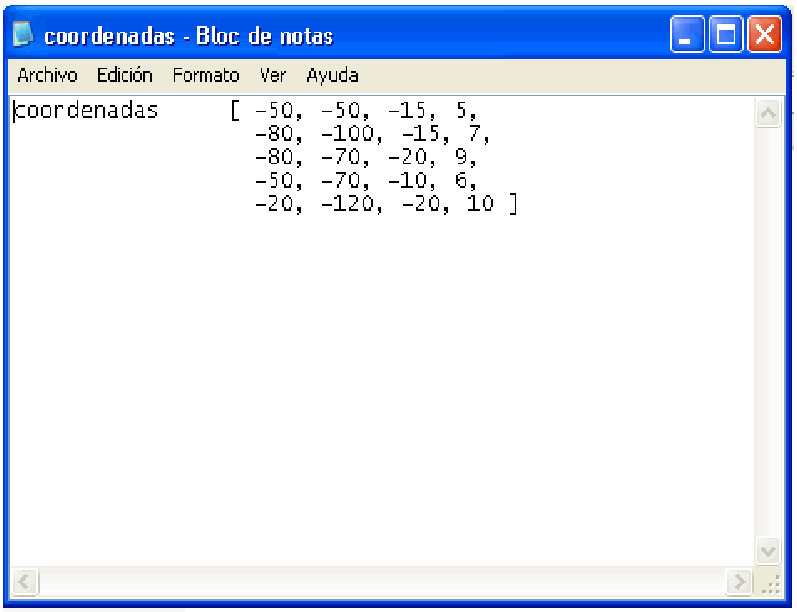
Se puede jugar de diferentes formas:

1.Simulación real: En este modo el jugador controla el helicóptero sin ningún tipo de restricción. Podrá volar por los diferentes escenarios realizando las maniobras que crea oportunas.

2. Por objetivos: El jugador tendrá que llegar a diversos puntos del escenario. Una vez llegado al punto "objetivo" saldrá otro "objetivo" en el escenario y el jugador se tendrá que dirigir hasta ese nuevo lugar. El punto al cual se tiene que dirigir el helicóptero es un cuadrado de color rojo.(véase la siguiente figura).

Para cambiar las coordenadas de los objetivos habrá que hacerlo desde la GUI eligiendo uno de los tres archivos posibles: coordenadas1.txt, coordenadas2.txt o coordenadas3.txt

Se puede editar la posición de los objetivos si se cambia el archivo coordenadas.txt. Este archivo tiene que tener un formato adecuado. Cada línea representa un objetivo distinto. Cada línea tiene 4 elementos separados por comas. Las tres primeras son las coordenadas de la posición del objetivo y la cuarta es la velocidad con que el helicóptero se dirige al punto. Todo esto se puede cambiar por el usuario pero siempre respetando el formato del archivo.



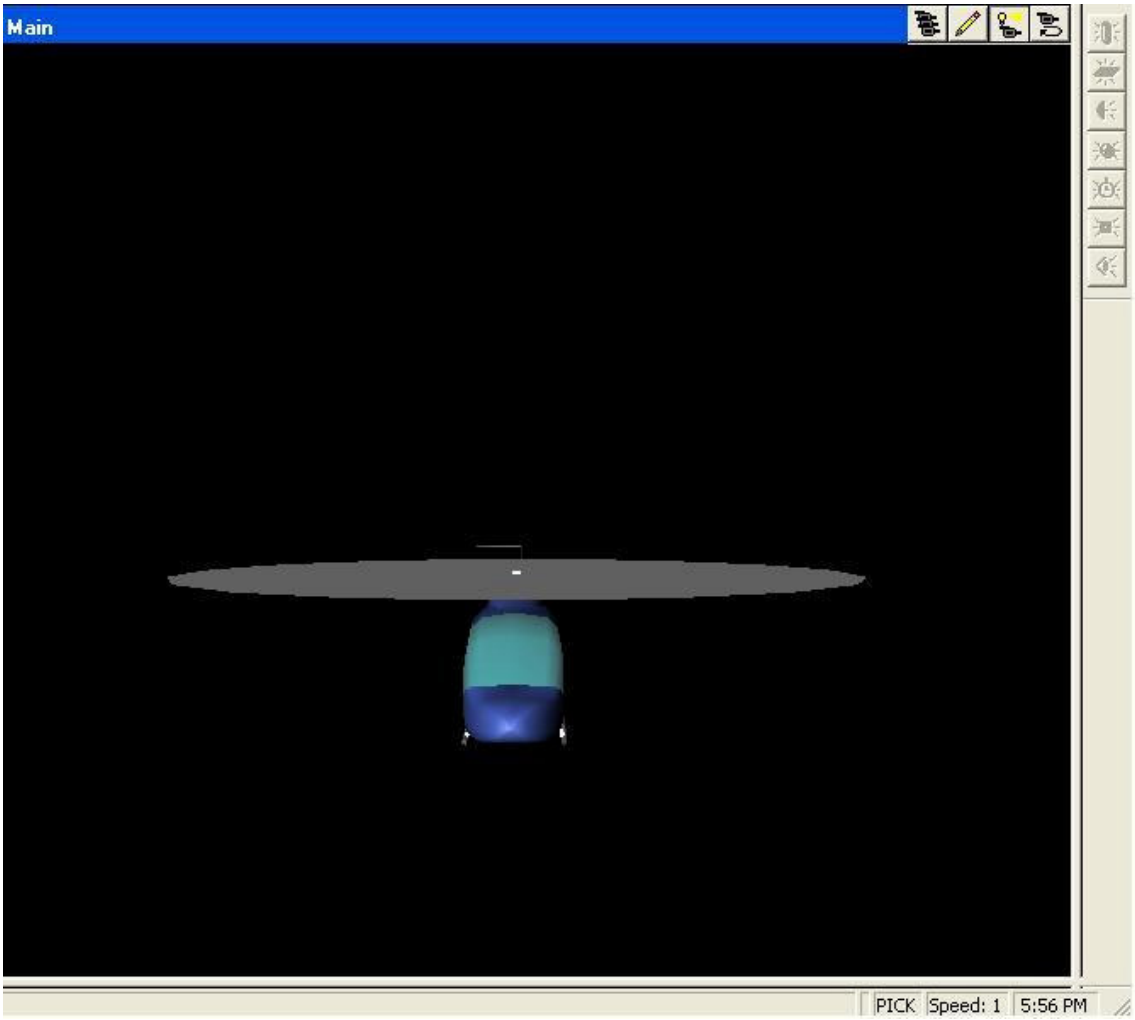
6 IMPLEMENTACIÓN

6.1 MUNDO VIRTUAL

6.1.1 HELICÓPTERO

El helicóptero ha sido importado de la librería de objetos del editor VRML. Es el objeto que se encuentra en la categoría Transportation(Air) denominado Helicopter(Lynx).

El helicóptero sería el siguiente:



Las medidas del helicóptero, serían:

- Ancho:2.3
- Alto:2.5
- Largo:8.6

El editor tiene un atributo denominado translation que devuelve la posición del helicóptero en el mundo. Esta posición no es la del centro del helicóptero, por eso al calcular las medidas respecto esta posición no es proporcional un lado del otro.

Las medidas respecto de la posición:

- Anchos: -1.3 y 1
- Alto:-1.3 y 1.2
- Largos:-4.8 y 3.8

De esta manera el centro del helicóptero tendría las siguientes coordenadas respecto de la posición que nos da el nodo translation:

(-0.15,-0.2,-0.5)

Estas coordenadas serán útiles para el cálculo de colisiones que se verá más adelante.

Otro de los atributos que cabe resaltar, es el de rotation, que gira el helicóptero en uno de sus ejes x, y o z según indica un parámetro llamado rotation.

6.1.2 ESCENARIOS

Para la creación de los diferentes escenarios usados en el proyecto se ha utilizado el editor VRML.

Hemos seguido los siguientes pasos:

- Construcción de los 9 terrenos: formando una matriz de 3x3, a su vez cada terreno individual esta formado por 100x100 unidades mínimas.

- Elaboración del relieve de cada terreno por medio del uso de la herramienta ELEVATION GRID. Con esto podemos ir subiendo o bajando el nivel del relieve y así construyendo las diferentes cordilleras, montañas y colinas en el caso de subida del terreno y de lagos y valles en el caso del descenso del terreno.

- Aplicaciones de los elementos MATERIAL y TEXTURE para dar el aspecto requerido al terreno. Así hemos elaborado terrenos llanos, rocosos y arenosos dando diferentes aspectos a cada uno de los escenarios.

- Introducción de diferentes elementos de decoración a nuestros mundos. Estos son elementos muy variados, desde diferentes tipos de casas o edificios hasta animales, árboles o aviones.

- Asignación del color del terreno. Así hemos elegido dar tres diferentes estilos para que cada escenario sea diferente uno del otro.

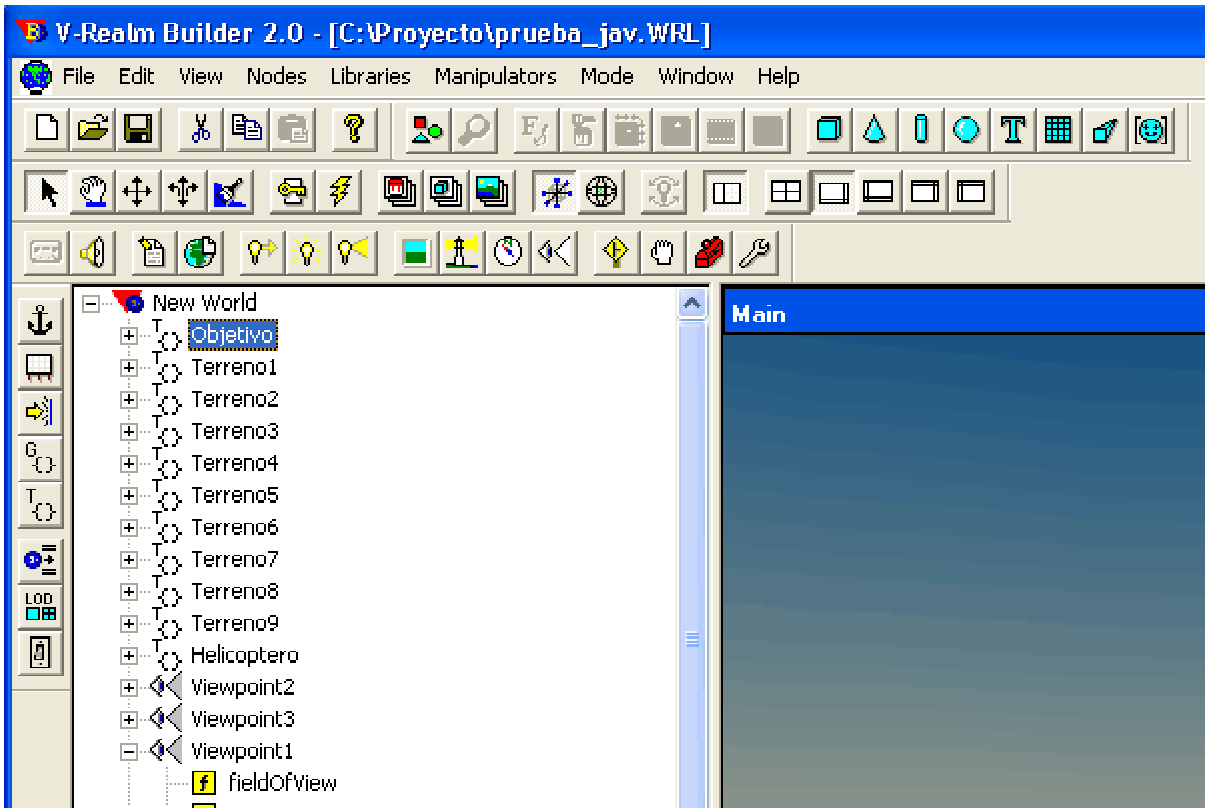
- Asignación del color a diferentes elementos del terreno como puede ser el azul en los lagos para simular el agua, el blanco a las cumbres de la montañas o el color propio de los diferentes elementos decorativos.

Todos los escenarios tienen que tener unos patrones de diseño comunes para que así el bloque VR Sink de Simulink nos los reconozca y no tengamos que cambiar el modelo de Simulink cada vez que queramos pasar de un escenario a otro distinto.

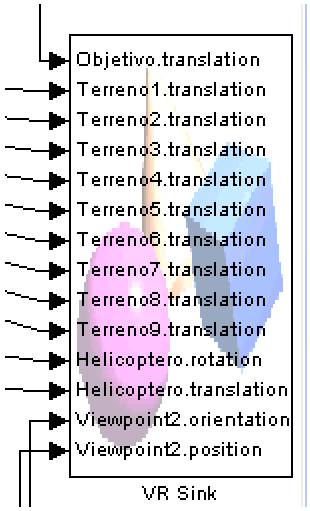
Estos patrones son los siguientes:

- Nodo Objetivo
- 9 nodos de terrenos (desde Terreno1 hasta Terreno 9)
- Nodo del helicóptero
- 3 nodos de vistas (desde Viewpoint1 hasta Viewpoint 3)

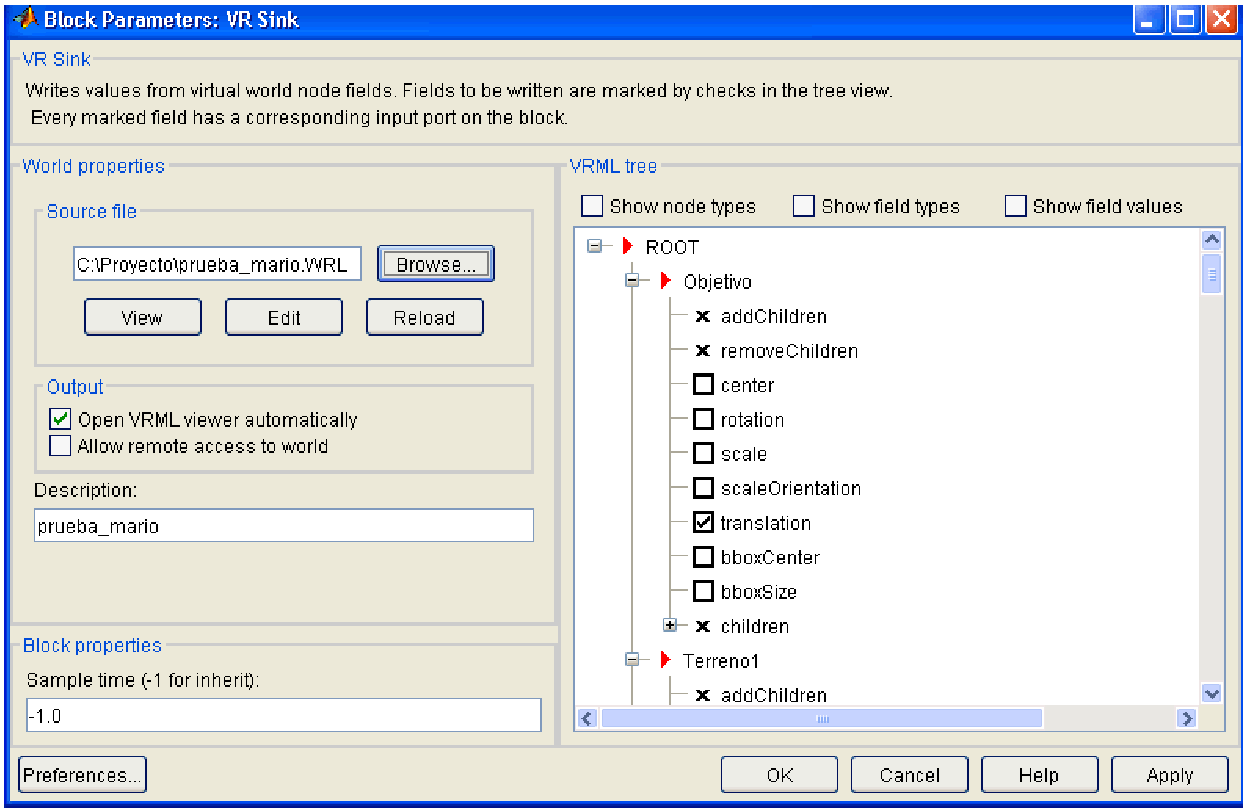
Por tanto los escenarios tienen que tener una estructura en el editor de la siguiente forma.



Ahora con estos patrones de diseño ya nos ajustamos a la características pedidas por el bloque VR Sink.



Y así ya podremos cambiar sin ningún problema el escenario actual e introducir el que queramos. Para esto habrá que hacer doble clic en el Vr Sink y cambiar la ruta que nos parece en el source file.



Hemos construidos tres escenarios diferentes, cada uno con sus propias características y singularidades.

6.1.2.1 Prueba_mario

Este escenario fue el primero que se construyó y sobre el que se ha realizado todas las pruebas que han tenido lugar en el desarrollo de la aplicación.

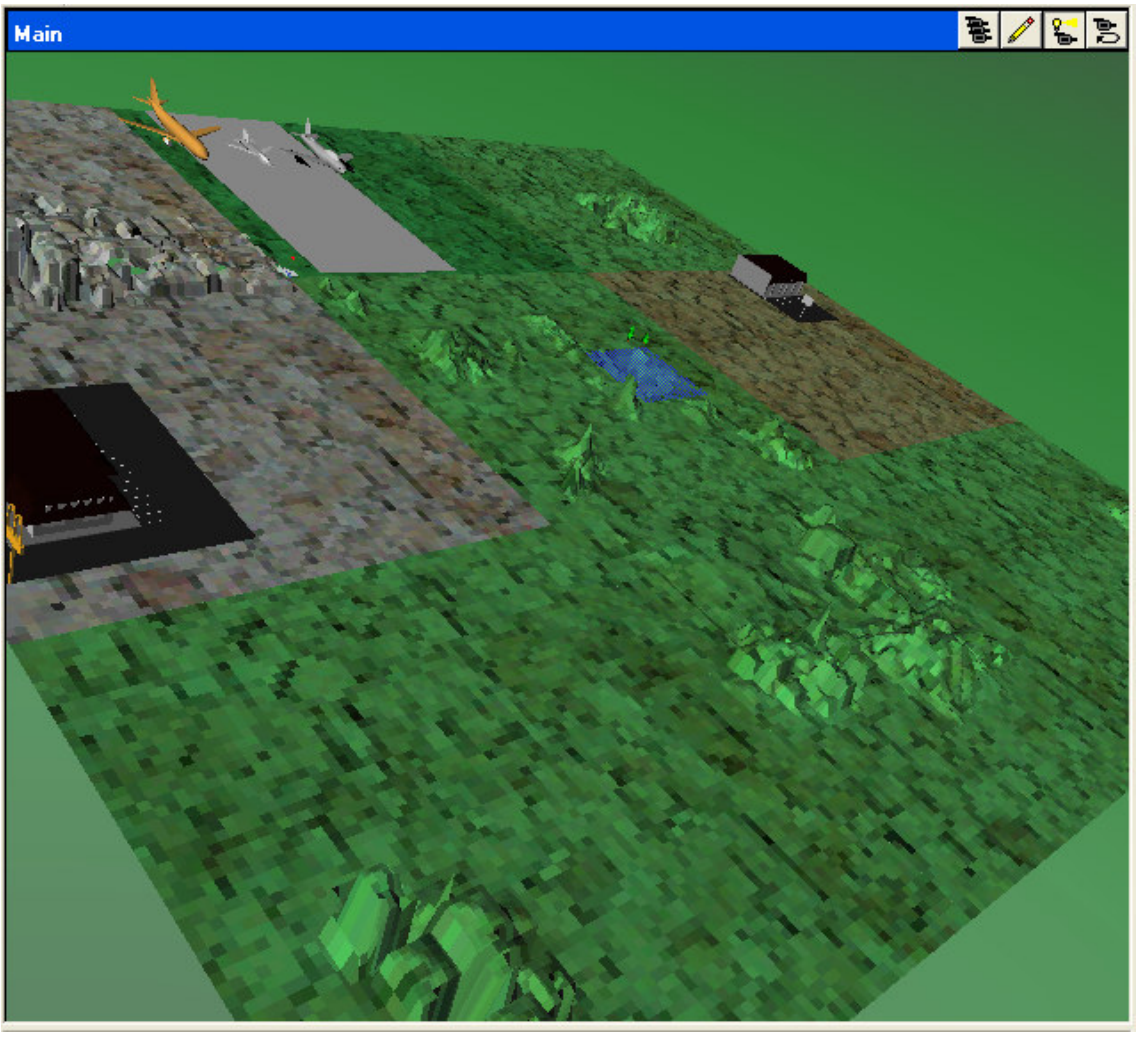
Tiene la particularidad de ser el más llano de los tres a excepción del terreno número 5 (el terreno de salida del helicóptero).En el terreno 5 también nos encontramos con un pequeño lago.

Además posee dos elementos decorativos como son la granja situada en el terreno 7 y una casa situada en el terreno 9.

6.1.2.2 Prueba_jav

Este escenario tiene la característica de ser el más montañoso de los tres. Se le ha dado un aspecto rocoso que acentúa esta propiedad. Se han introducido dos cadenas montañosas importantes, una en el terreno numero 8 y otra situada entre los terrenos número 1 y 4.

Entre los elementos decorativos cabe destacar los dos edificios situado en los terrenos 4 y 6 respectivamente. Además en el terreno 2 hay una pista de aterrizaje con tres aviones, simulando un aeropuerto.



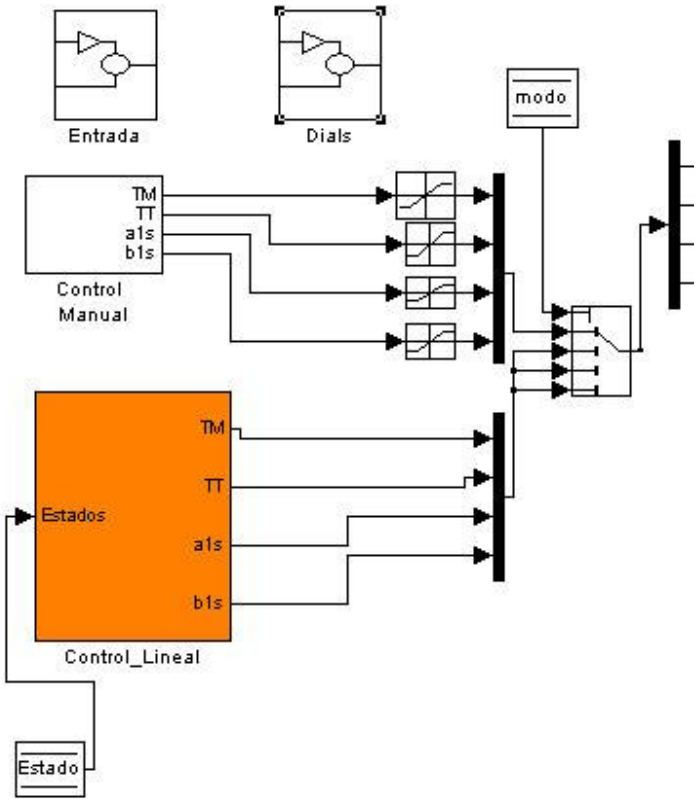
6.1.2.3 Prueba_ivan

Al último de los escenarios se le dio un aspecto arenoso para simular unas condiciones del terreno que fueran las de un clima seco.

Aparecen dos pequeños lagos o oasis además de dos casas situadas en los terrenos 1 y 9. También hay una granja en el terreno 7 en la que se pueden encontrar diferentes animales(cerdo, pato, conejo, burro...).

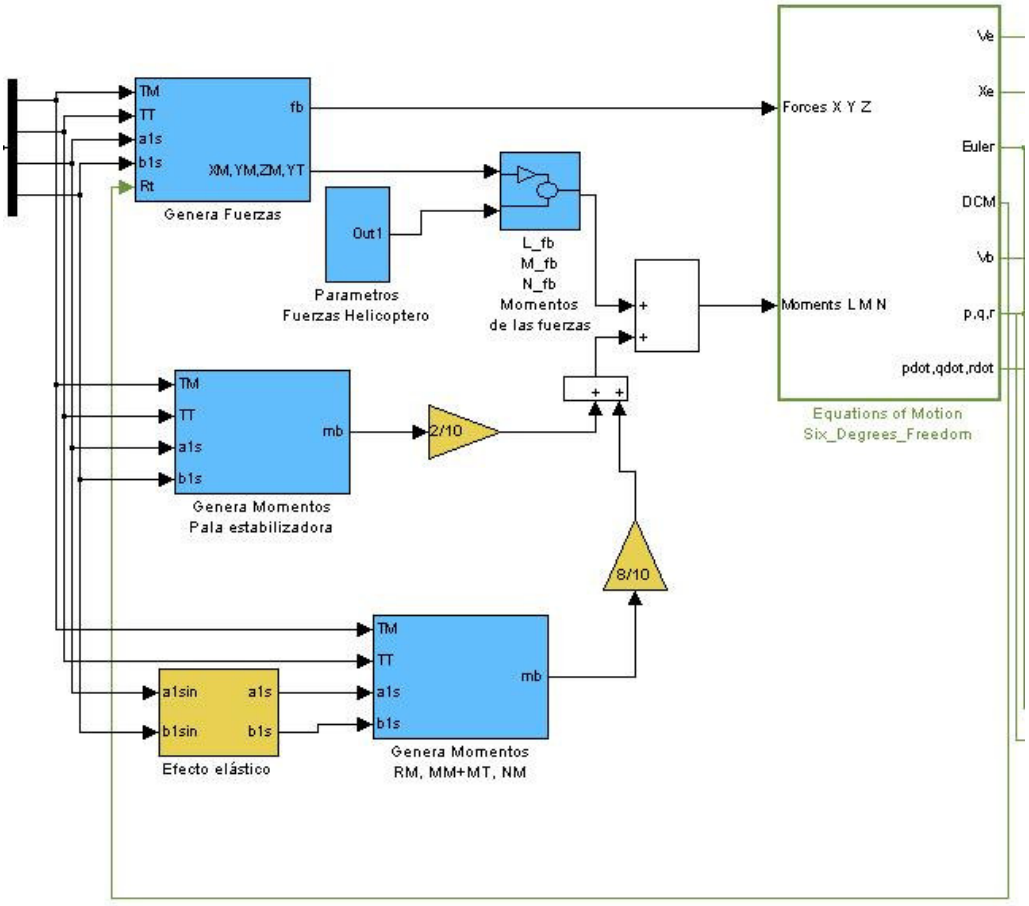
6.2 MODELO SIMULINK

Ahora hablaremos de la implementación de la parte de Matlab y Simulink de nuestro simulador. Para poder entender bien el funcionamiento de modelo, y sobretodo para tener una pequeña base para entender las futuras explicaciones, vamos a hacer ahora una breve descripción del funcionamiento general del sistema. Separando todo nuestro bloque Simulink en 4 partes, esto nos permitirá hacernos una idea global del sistema.



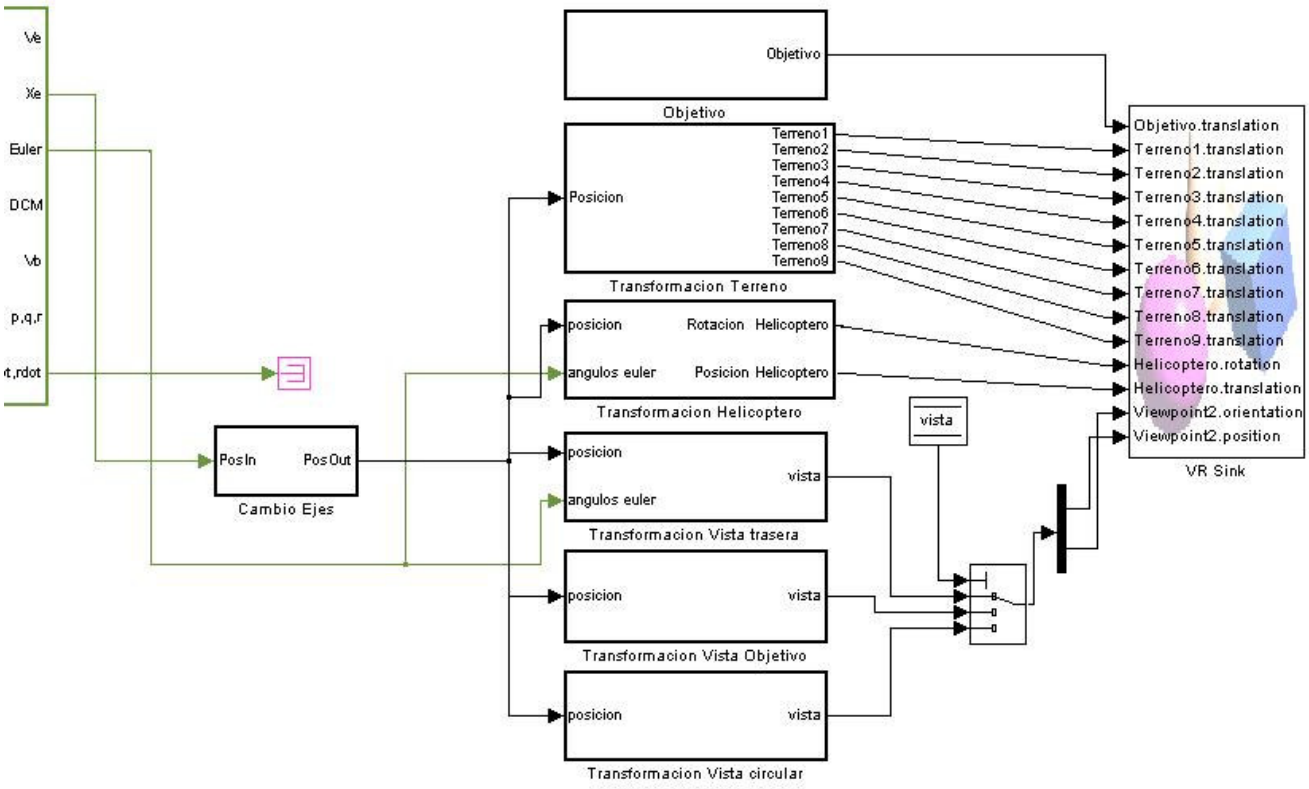
En primer lugar hablaremos de esta parte del sistema, e intentaremos ver como sería un ciclo de reloj entero. Primero se capturaría la entrada, en el bloque con mismo nombre. Entradas para las diferentes variables, el modo de uso, o el movimiento de la palanca de mandos. Ese bloque se encuentra a parte, puesto que cambia variables que se usan en todo el modelo, y que se guardan en bloques del tipo "*Data Store Memory*", como los que vemos en la imagen, para el Estado y el Modo. Luego están las dos formas de controlar el helicóptero, el control manual y el piloto automático, que serían los bloques "*Control Manual*" y "*Control_Lineal*" respectivamente. El piloto automático se basa en el estado anterior del helicóptero.

Ambos modos lo que hacen es dar valores a las variables que controlan el helicóptero, TM, TT, a_{1s} y b_{1s}. Para acabar lo que hace es elegir cual de las opciones de control manda a la siguiente etapa. Al final de este punto tenemos las 4 variables que determinarán las fuerzas y momentos que se generan en cada instante. Todo esto se vera mejor en los subapartados 2 y 3.

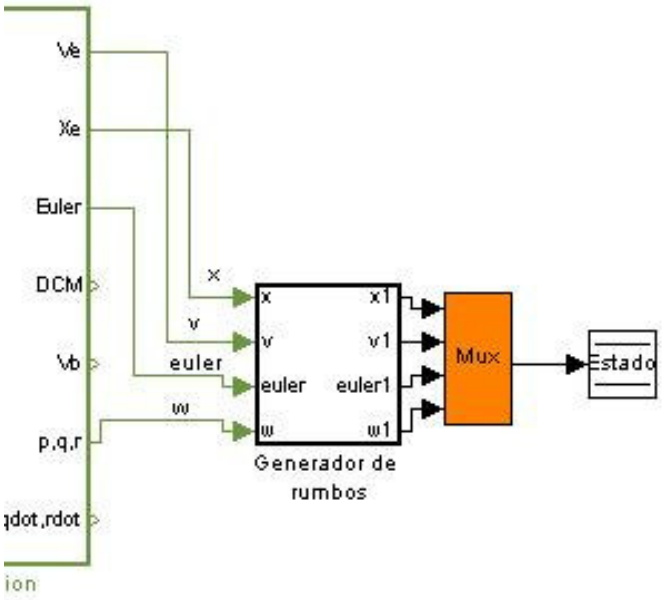


Esta sería la segunda etapa. En esta etapa, con las 4 variables que llegan de la anterior, se definirán las fuerzas y momentos que marcaran el siguiente estado del helicóptero. Esta es la parte del modelo que teníamos al empezar. Para una descripción más profunda que la que ya ha habido hasta ahora, véase el documento "Modelación del sistema no lineal de un helicóptero", cuyos datos se encuentran en la bibliografía. El bloque superior genera las fuerzas, y los dos bloques azules inferiores, los momentos. Uno corresponde al efecto de los rotores de cola y principal, y el de en medio al efecto de una pala estabilizadora, aportada por Guillermo Martínez como colaborador en nuestro proyecto.

El bloque verde llamado, "*Equations of Motion Six_Degrees_Freedom*" es el que procesa esas fuerzas y momentos para devolver el estado del helicóptero, definido sobretodo por la velocidad, posición, ángulos de euler y velocidades angulares. En este bloque se controlaran también las colisiones y se sacaran los datos para los diales. Se explicará mejor en los subapartados 5 y 6.



Esta tercera parte, sería en la que mostramos los resultados, para mostrarlos solo necesitamos la posición y los ángulos de euler actuales de helicóptero, resultado de las 2 etapas anteriores. Aquí es donde se hacen todas las transformaciones necesarias y se sacan los datos requeridos por el mundo virtual. Se tratará más a fondo en los subapartados 7,8,9 y 10.



En esta cuarta parte, que se podría suponer que se ejecuta más o menos a la vez que la anterior, es cuando se prepara el estado siguiente que se le va a pasar al piloto automático. Decimos bien, se prepara, puesto que es aquí, donde se le dirá al piloto automático que es lo que queremos que haga. Se verá más en detalle en subapartado 4.

Esta es más o menos la idea en general del funcionamiento del programa, se generan unas variables de control, que generan unas fuerzas y momentos, que crean un estado, que se muestra y se guarda (modificado o no), para la siguiente generación de variables de control. Ahora veremos todas las etapas detalladamente.

6.2.1 INICIO

Aquí hablaremos de todo lo que hay que hacer antes de empezar a usar el modelo. Y sobretodo de lo que hacen los ficheros Matlab que se ejecutan al inicio.

El primero en llamarse, es el script "*Lineal_Heli.m*", que a su vez usa otro script llamado "*Ecuacion_Estado.m*". Estos dos ficheros son con los que contábamos antes de empezar el proyecto. Su papel es definir todas las variables y constantes del modelo. Por un lado las variables que definen el helicóptero, como la distancia desde el rotor de cola al centro de

gravedad... Definen todo lo que es la física del helicóptero, ecuaciones de fuerzas y momentos sobre el helicóptero y sobretodo realiza el trimado del helicóptero con esos valores que lo definen. De esta forma consigue los valores de TM, TT, a1s y b1s para los que el helicóptero esta en equilibrio.

Para más detalle, referirse a la memoria de proyecto "Modelación del sistema no lineal de un helicóptero" de Guillermo Martínez Sánchez.

A partir de aquí ya tenemos la física, y quedan definir todas las variables necesarias, para el entorno y control de este simulador. Esto se hace con el script "*Inicio.m*". En el se declaran todas las variables globales que se van a utilizar, y se inicializan. Aquí definimos también las variables que usa el bloque "*Equations of Motion Six_Degrees_Freedom*", si se usara el bloque original de Matlab estas variables se inicializarían, con un doble click en el bloque, pero para hacer las colisiones tuvimos que realizarlo nosotros mismos, y ahora hay que realizar esas modificaciones en el script "*Inicio.m*".

Aquí también se inicializa el mapa de terrenos, así como el mapa de colisiones, teniendo en cuenta el terreno en el que nos encontramos, definido en la variable "*actual*". Por último se lee de fichero el camino de objetivos que queremos que siga nuestro helicóptero.

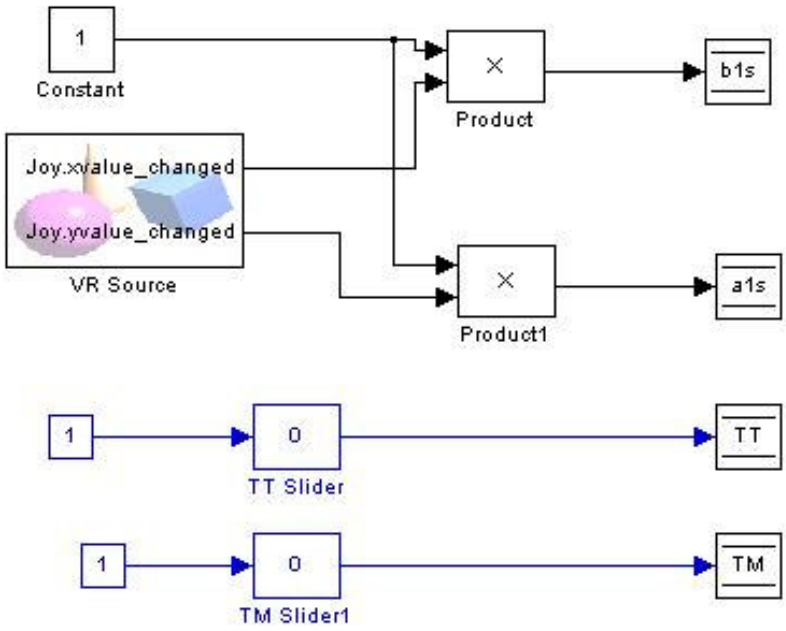
Tras este fichero se llama al último script, "*DefineMulle.m*", Aquí es donde se definen las variables que usara la pala estabilizadora. Las dos variables más importantes de este script, puesto que son las que se pueden modificar para llegar a un mejor funcionamiento son:

- ***Kmuelle***: Define la elasticidad del efecto muelle de la pala estabilizadora.
- ***Nurozamiento***: Define el rozamiento de la pala, cuanto más alto sea, más tardara en reaccionar, y en hacer los movimientos adecuados.

6.2.2 ENTRADA

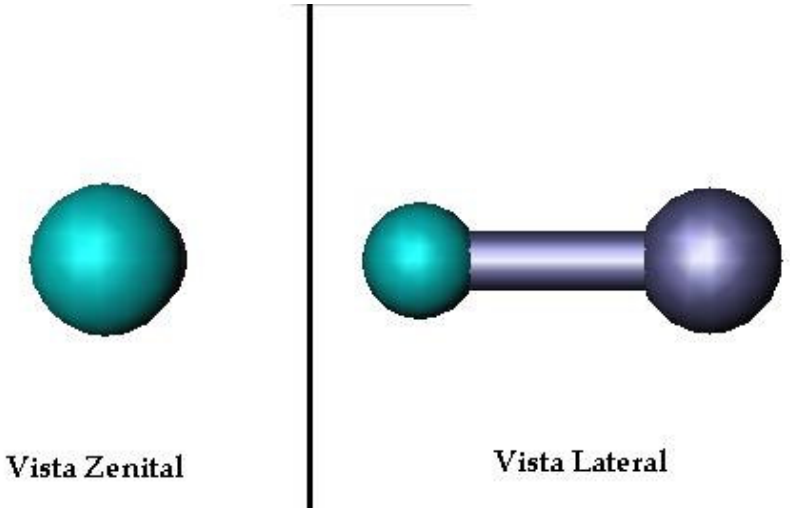
Explicaremos ahora los métodos que tenemos para que el usuario interactúe con el simulador. Sobretodo se basa en que tenemos dos modelos, uno en el que se usa un joystick virtual, y otro en el que se usa el teclado. Como se explicará luego, veremos que ambos métodos era incompatible tenerlos en un mismo modelo.

6.2.2.1 Entrada con Joystick Virtual



Este es el método que más hemos usado durante el año, puesto que la forma de poder usar el teclado no la descubrimos hasta muy avanzado el proyecto.

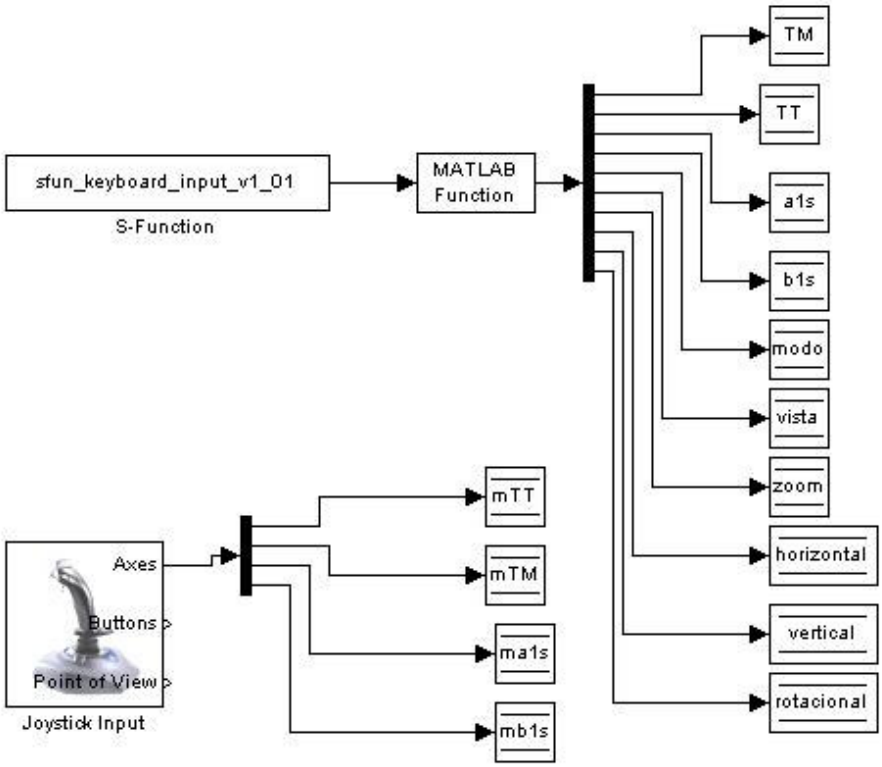
Este joystick, lo implementamos en un mundo virtual, con dos esferas y un cilindro.



Para poder usarlo en Simulink, al contrario que el mundo, que es como salida, este debe ser como entrada, para esto se usa el bloque VRSource. Como todo Joystick tiene dos grados de libertad, y la bola verde, tiene siempre una posición [x, y]. Estas coordenadas son las que usamos para dar valor a la inclinación del rotor principal (a1s y b1s). Esos valores se mueven como en casi todos los joystick entre 1 y –1 en cualquiera de los dos ejes. Por eso luego es necesario hacer una transformación a ese valor.

Como se puede ver también, en este modelo el resto de valores, como TM y TT principalmente, por no hablar de la selección de modos y los valores de la vista circular, se deben tomar gracias a Sliders. Esto hace que sea mucho más incómodo este modelo, que el que ya incluye al teclado. Pero como ambos eran incompatibles, decidimos tener dos modelos diferentes.

6.2.2.2 Entrada con Teclado



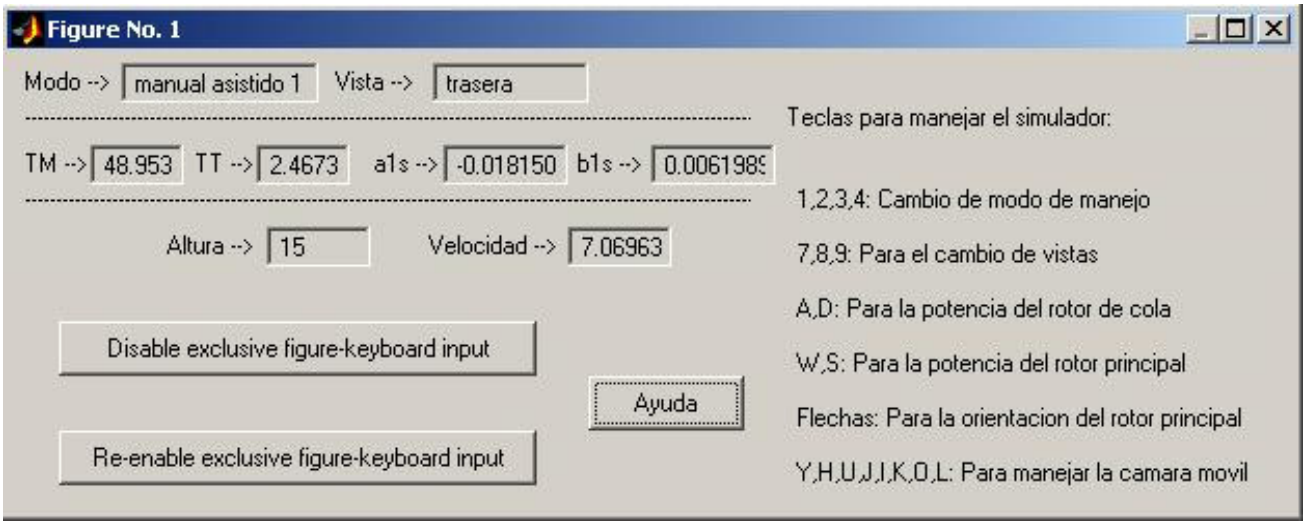
A primera vista se puede observar con respecto al modo anterior, que este controla todas las variables del programa. Tienes mucho más control, y más centralizado que con el joystick virtual. Aquí todo lo hacen funciones de Matlab, una, la que captura las entradas, la S-Function "sfun_keyboard_input_v1_01", y otra, la que trata esas entradas, la función "teclas". La primera es sin duda la más complicada de las dos.

Funciona de esta forma. Crea una "figure", que es la que captura las teclas, es la única forma que encontramos de conseguir capturar la entrada por teclado. Pero esta opción tiene 2 problemas:

- Que esta pantalla debe estar siempre activa, y por delante de las demás.
- Que no usa un interrupción estilo KeyPressed, de C++, sino que te devuelve la tecla pulsada actualmente.

El primero de ambos problemas, causa que sea incompatible con el joystick virtual, puesto que esta debe estar siempre por encima. El segundo hace que cada vez que pulses una tecla te la devuelva un numero diferente de veces, según el tiempo que la pulses tu, y el tiempo actual de cada instante de simulación, que esta en paso variable. Eso causa que el modo manual con teclado no sea muy preciso.

La ventana que se crea para capturar la entrada por teclado, puesto que debe estar activa, la hemos usado para mostrar datos importantes, como el modo de uso actual, la velocidad, la altura...



Esta función tiene dos partes importantes, "*mdlInitializeSizes*" que es donde se crean las paneles para los textos, los botones y la ventana en general. Aquí vemos como se crea la ventana principal.

```
handle.figure=findobj('type','figure','Tag','keyboard input figure');

handle.figure=figure('position',[10 40 370 220],...
                     'WindowStyle','Modal',...
                     'Color',get(0,'DefaultUicontrolBackgroundColor'));

set(handle.figure,'Tag','keyboard input figure');
```

El parámetro WindowStyle a Modal es lo que hace que solo puedas pinchar con el ratón en esta ventana. Cuando pulsas el botón “Disable exclusive...” pasas este parámetro a Normal, y entonces ya puedes tocar en el resto.

Para que capture la entradas de teclado debe ser la ventana la que esté seleccionada. Si pulsas por ejemplo en ayuda, pulsa luego otra vez en la ventana, para que sea esta la seleccionada. Como vemos en la imagen de la ventana, ahora no capturaría las entradas de teclado, puesto que es el botón “ayuda” el que está seleccionado.

La otra parte importante es el “*mdlUpdate*”, que es donde captura la entrada de teclado, y donde se actualizan los textos y valores de los diferentes paneles.

El valor que devuelve la función es el valor ASCII de la tecla pulsada. En caso de que no haya nada pulsado devuelve el valor 43, por esto no se puede usar la tecla ‘+’.

Ahora ya tenemos el valor pulsado, una vez con esto, se manda ese valor a la función “teclas”, que lo trata. Esta función, basándose en las variables globales y en la tecla pulsada, devuelve la salida:

```
Salida = [cTM, cTT, ca1s, cb1s, modo, vista, zoom, horizontal, vertical, rotacional1]
```

Es decir, todos los valores necesarios, en principio la tecla pulsada solo va a cambiar el valor de alguno de estos valores, para el resto devolverá el antiguo valor, vemos aquí un ejemplo de cómo trata el caso en el que pulsas la tecla ‘a’.

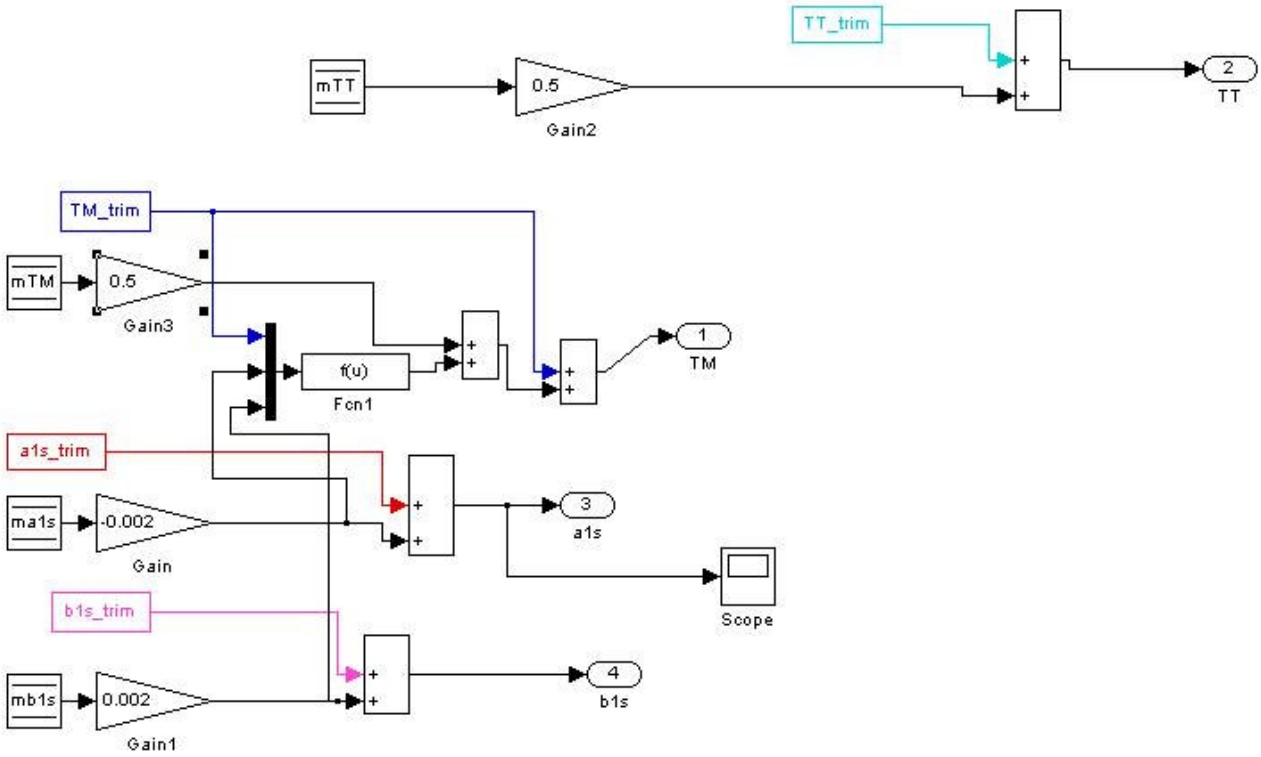
```
%tecla "A" --> girar izq.
case 97
  ca1s = 0;
  cTM = 0;
  cTT = cTT - 2;
  cb1s = 0;
```

Por esa incompatibilidad con el joystick virtual, y debido a la poca precisión del teclado, para el modo manual nos hacía falta otro tipo de control. Para este modo de uso, se usó el bloque "*Joystick input*", a pesar de que no queríamos obligar al usuario a tener un joystick. De esta forma tenemos un control tan exacto como el que teníamos con el joystick virtual en el otro modelo.

Dependiendo del tipo de joystick que se tenga, se deberá trabajar con las salidas del bloque "axes" o con la de "buttons". En nuestro caso, como era un mando de "*Playstation2*", hemos trabajado con las dos ruedas, cuyas respuestas las captura el bloque en "axes". Pero si se tiene una sola palanca, como es habitual, lo mejor seria usar los botones para las variables TM y TT. Esto el bloque lo devolvería con señales de binarias, en la salida "buttons", según este el botón pulsado o no.

Las variables de Simulink TM, TT, a1s y b1s, se usarán en este modelo solo para los modos manuales asistidos, y para el manual se usaran las variables mTM, mTT, ma1s y mb1s que provienen del joystick.

6.2.3 CONTROL MANUAL



El control manual funciona de una forma muy sencilla. Tenemos los valores de trimado, para los que el helicóptero esta estable. Lo que manejamos con el control manual es una desviación con respecto a esos valores.

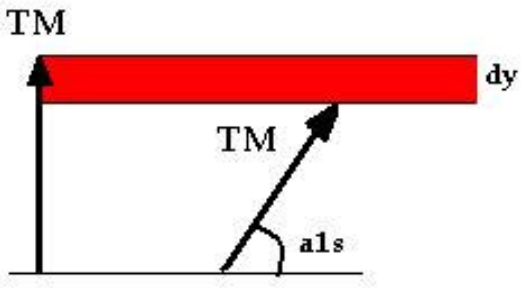
Podemos distinguir dos partes, el control de la variables a1s, b1s y TT, y por otro lado el control de la TM. El control de las primeras funciona de esta forma:

Ent x Ganancia + ValTrim = Salida

Tan solo varía la ganancia, para los ángulos es de 0.002 y para la TT es de 0.5.

En cuanto al segundo caso, el control de la TM, hemos introducido una pequeña ayuda, intentando hacer que el control sea más fácil. La idea es disminuir el efecto negativo que se

produce al girar el rotor principal del helicóptero. Al girar aumentas el ángulo de inclinación de la fuerza TM, y por tanto al mantener la misma fuerza del rotor, su componente 'Y', será menor, y el helicóptero irá cayendo. Como uno de los mayores problemas que hemos tenido durante el año, ha sido el de poder controlar todas la variables a la vez, era necesario, sino corregir este efecto, hacerlo menor. En la siguiente imagen, vemos ese efecto en 2 dimensiones, que sería igual para 3 dimensiones.



Es la 'dy' la que intentamos corregir. Por esto, introdujimos una pequeña ayuda a la TM, cuando la a1s y b1s aumentaran, aumentaría la TM en consonancia para no perder altura. Esta pequeña ayuda, parecida al control de velocidad en los coches, ha sido útil, pero no resolvió por completo esta progresiva caída al moverse, puesto que la interacción mutua, de estas cuatro variables, es mucho más compleja a la hora de crear las fuerzas que generan en el helicóptero.

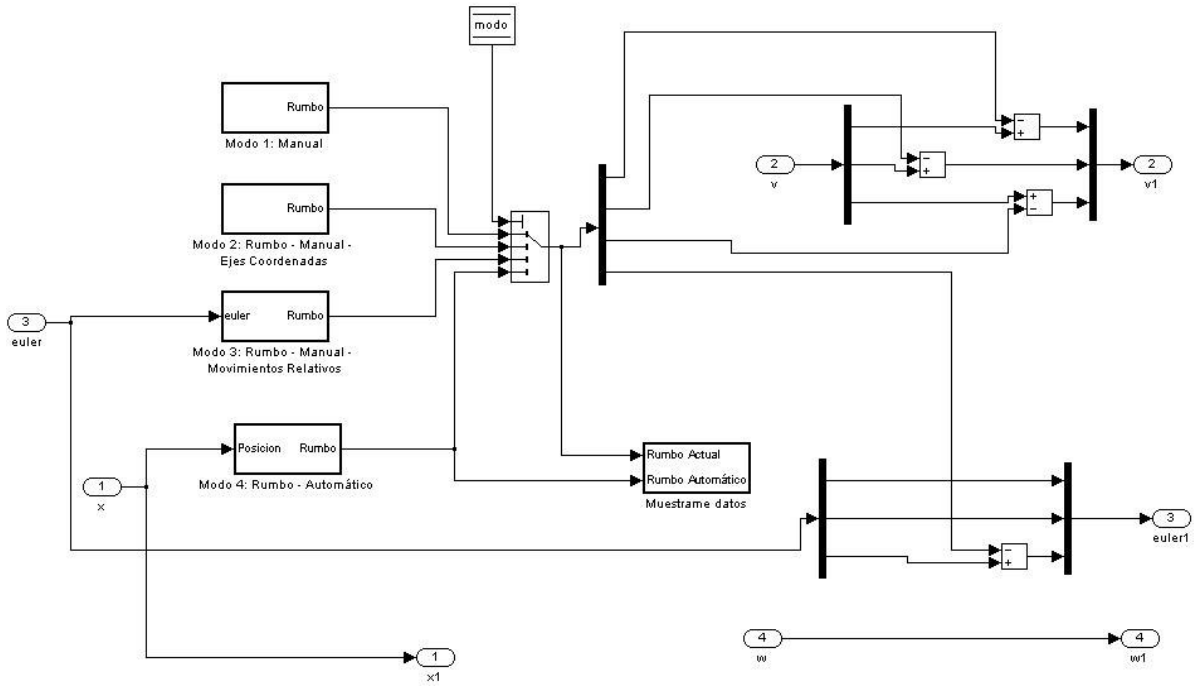
El control de TM queda de esta forma:

Ent* x *Ganancia* + *ValTrim* + ((49 / (cos (*a1s*) x *cos(b1s*))) – 49) = *Salida

La decisión de usar como valor 49, en vez de TM_trim, se debe a un mejor funcionamiento con el primero que con el segundo, aunque lo lógico hubiera sido lo segundo.

Para los dos modelos, el de teclado y el del joystick virtual, el funcionamiento es el mismo. Cambian solo las ganancias, que son el resultado de muchas pruebas.

6.2.4 RUMBOS



En esta apartado hablaremos del módulo "Generador de rumbos", que lo que hace es indicarle al piloto automático (módulo Control_Lineal), la velocidad que debe tomar en cada eje, y en algunos casos, con que yaw debe ir.

Como ya se ha explicado anteriormente, el módulo Control_Lineal, consigue que el helicóptero se quede estable en la posición en la que esté. Por esto, para conseguir que dicho módulo haga que el helicóptero vaya a 5 m/s hacia delante, debemos engañarle diciéndole que va a –5 m/s, así cuando lo intente estabilizar, lo pondrá a 5 m/s.

Por esto, los valores que devuelven los módulos de los diferentes tipos de rumbos, se restan a los valores actuales de la velocidad del helicóptero. A su vez, esto también explica el porque de la colocación de este módulo donde está, que es justo después del bloque *"Equations of Motion Six_Degrees_Freedom"*, que devuelve los estados del helicóptero, y antes de que estos estados le lleguen al bloque "*Control_Lineal*", para poder alterarlos en caso de necesidad (alguno de los modos de manejo por rumbos).

Hay cuatro tipos de manejo:

- ***Manual:*** Es el modo 1, el helicóptero se mueve según las entradas manuales, en ese caso, este módulo no debe hacer nada, por lo que los estados que le lleguen al piloto automático serán exactamente los que tenga el helicóptero. Aunque claro está, los valores de TM, TT, a1s y b1s que devuelva el piloto automático, en este modo no se usara, puesto que el que estará dando esos valores seremos nosotros mismos, como vimos en el apartado "*Control Manual*".
- ***Rumbos Manuales según Ejes:*** Este corresponde al modo 2, de los dos modos de rumbos manual, es el que mejor funciona, pero todo esto lo explicaremos en detalle en un posterior apartado.
- ***Rumbos Manuales según orientación:*** Es el modo 3, y al que el anterior, lo explicaremos luego.
- ***Rumbos Automáticos:*** Este es el modo 4, y hace todo automáticamente, irá hacia los objetivos designados por fichero. Luego lo veremos con más detalle.

Para la selección de un modo u otro, se hace con dos switches, controlados por la variable modo. Uno colocado aquí, que controla los valores con los que se modificará la velocidad y el yaw del helicóptero, según que modo de pilotaje se esté usando. Y el otro está colocado a la salida del piloto automático, que a menos que el modo sea 1 (Manual) dará el control al piloto automático.

Una cosa muy importante en lo que se refiere a los rumbos es el concepto de objetivo. El objetivo es el punto al que se dirigirá el helicóptero en modo automático, una vez alcanzado un objetivo, este cambiará y se convertirá en otro.

La lectura de los objetivos, se hace al inicio (al cargar el modelo), como se explicó en punto de inicio. Esta serie de objetivos forman un ***camino***, que es el que seguirá el helicóptero. Una de las posibles metas de este simulador para el usuario puede ser el conseguir llegar a los

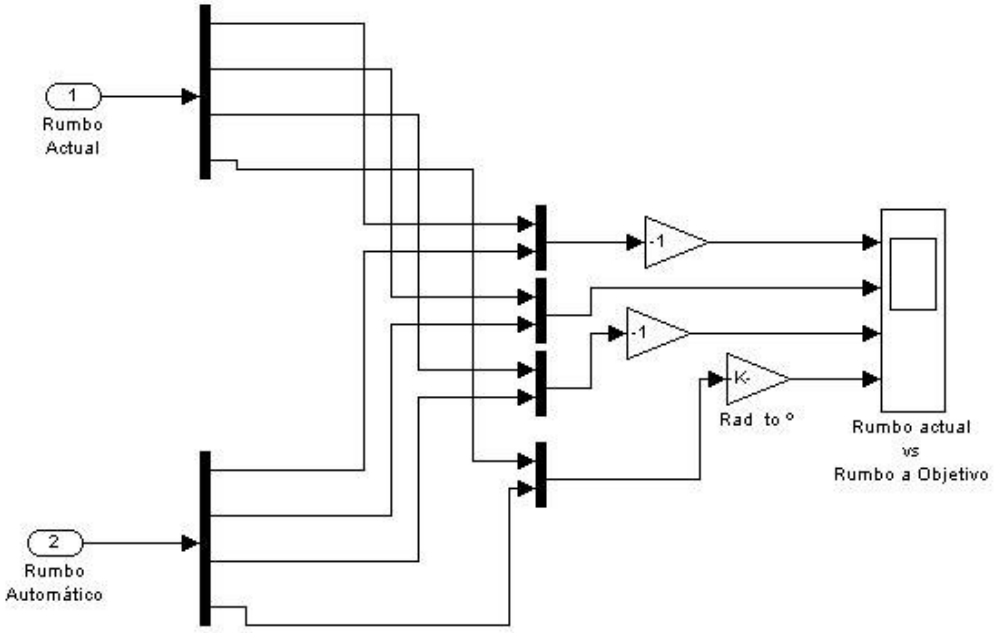
objetivos en cualquiera de los modos manuales. Para leer este camino, se usa la función "*calculavector1*" desde la que se lee del fichero "*coordenadas.txt*". Este fichero esta compuesto por las coordenadas de los puntos objetivo además de por la velocidad a la que se quiere que el helicóptero vaya hacia allí.

```
coordenadas     [ -50, -50, -15, 5,
                  -80, -100, -15, 7,
                  -80, -70, -20, 9,
                  -50, -70, -10, 6,
                  -20, -120, -20, 10 ]
```

6.2.4.1 Rumbos Manuales

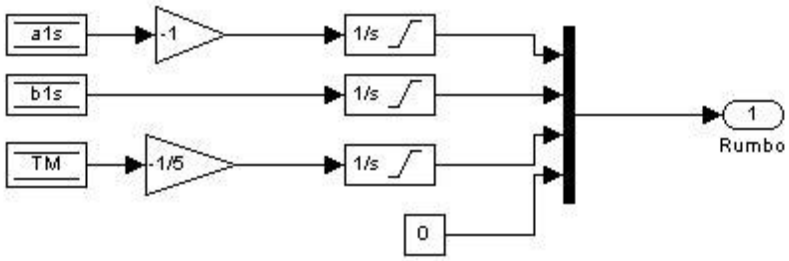
En estos modos a pesar de ser el piloto automático el que maneja, el usuario es el que controla lo que haga el piloto automático. Básicamente en el modo manual a secas, el usuario controla las variables TM, TT, a1s y b1s para hacer los movimientos que él desee. Mientras que en estos dos modos, el piloto automático se ocupa de esas variables pero para hacer lo que el usuario quiera.

Para facilitar el uso de estos modos, en lo que se refiere a llegar a los objetivos, hemos incluido un scope, en el bloque "*MuestrameDatos*".



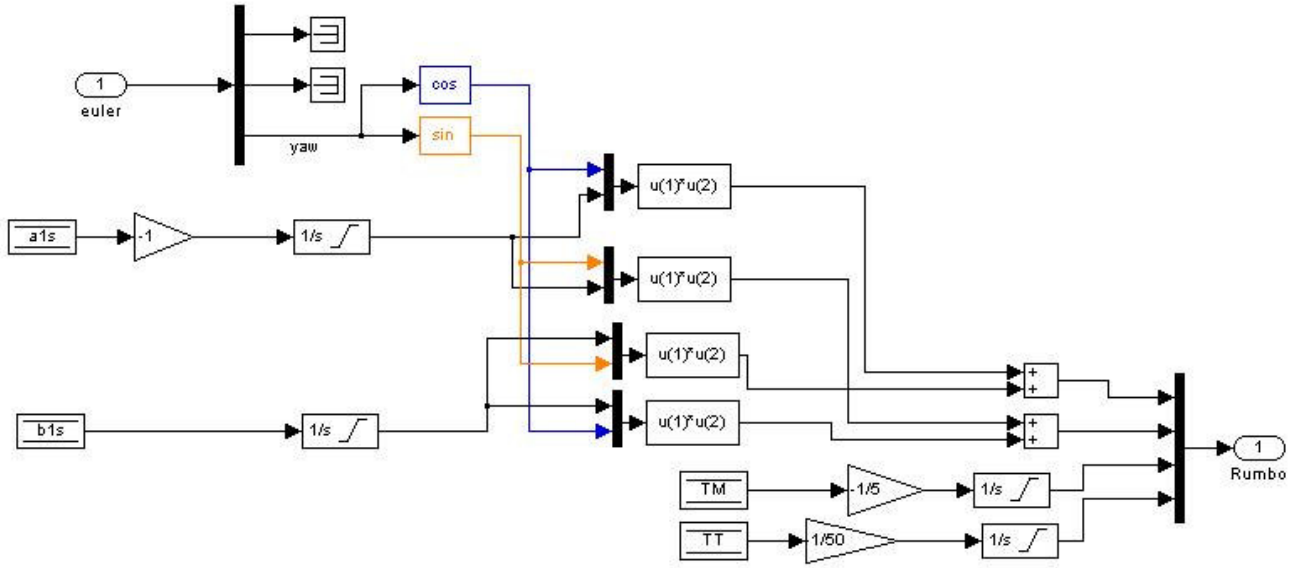
Este scope es una comparativa entre lo que haría el piloto automático si te pusieses en modo 4, y lo que actualmente le estas mandando al helicóptero que haga.

6.2.4.1.1 RUMBO MANUAL SEGÚN LOS EJES



Como vemos es bastante sencillo, 3 integradores recogen los valores de las variables globales de Simulink TM, a1s y b1s. Dándote de esta forma control sobre la velocidad querida en cualquier dirección de los 3 ejes. Por ejemplo con la a1s, manejas la velocidad en el eje de las z's.

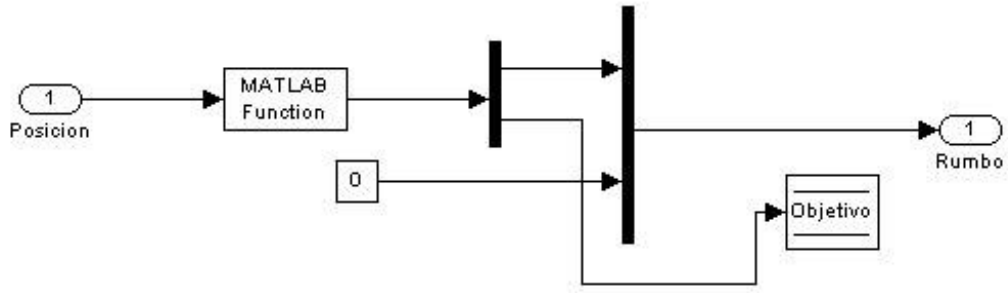
6.2.4.1.2 RUMBO MANUAL SEGÚN LA ORIENTACIÓN



Este modo con respecto al anterior te da más libertad, puesto que además de la velocidad en los 3 ejes, te permite manejar el yaw del helicóptero. Aunque esto es un arma de doble filo, puesto que te permite girar el helicóptero, pero el piloto automático usa el yaw para controlarlo, así que le estas quitando un grado de libertad y el funcionamiento es mucho peor, causando muchas inestabilidades.

Como se ve en el nombre de este apartado, este modo, no se basa en los ejes, sino en la orientación. Por esto el yaw del helicóptero influye en las velocidades en las X's y en las Z's. Ahora la a1s y la b1s ya no controlan el movimiento en las z's y las x's sino el movimiento hacia delante y hacia los lados.

6.2.4.2 Rumbo Automático



Este modo es el que le da total control al piloto automático, y va siguiendo el camino punto por punto, hasta acabarlo, cuando entonces vuelve a empezar.

El modo automático al igual que el manual según los ejes, también controla solo las velocidades en los 3 ejes y deja el yaw tal como llega. Todo el trabajo en este modo lo hace la función Matlab "*indicaCamino*". Cuyo trabajo es mirar a que punto del camino quieres ir, ver si has llegado, en ese caso pasaría al siguiente y en caso de no haber llegado, que da el vector de velocidades para llegar a él.

Ese vector de velocidades como se puede ver en la función "*dameRumbo*", utilizada por "*dameCamino*", es el vector desde la posición del helicóptero al objetivo, normalizado y luego multiplicado por la velocidad deseada.

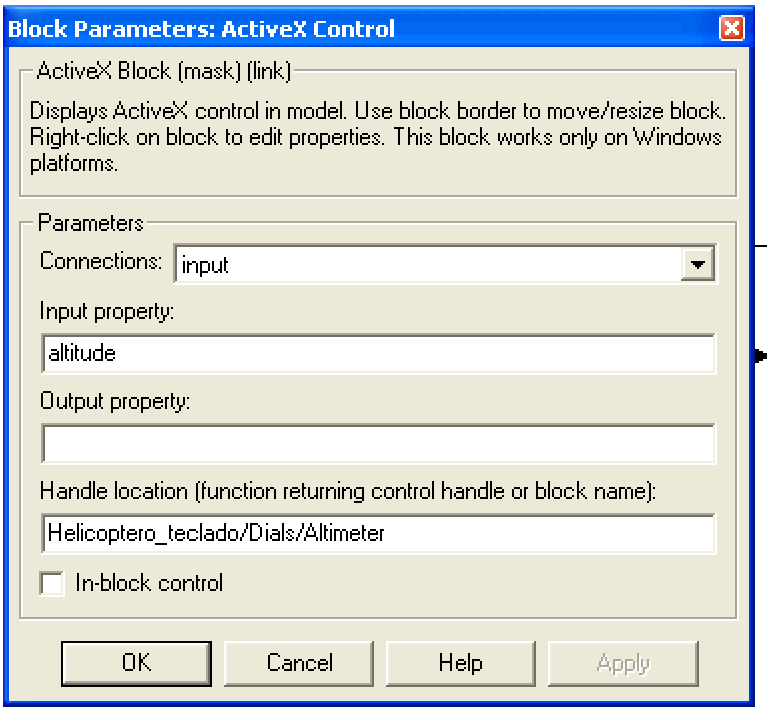
Este módulo usa las variables globales "puntoActual", "Objetivo" y "camino".

6.2.5 DIALES

Los diales que hemos utilizado para nuestro proyecto son los característicos entre los simuladores de aviones o helicópteros. Estos diales los encontramos en la librería de Simulink Dials and Gauges Blockset. Son muy importantes para intentar recrear un cuadro de mandos de un helicóptero. Hemos usado un altímetro, un velocímetro y el horizonte artificial.

Lo más importante en la realización de los diales ha sido la utilización del bloque ActiveX. Este bloque nos ha dado la posibilidad de tener nuestros diales en un subsistema de Simulink aparte y así poder dar la sensación de tener un cuadro de mandos. Sin este bloque la presentación de los diales por pantalla estaría ligada a los demás bloques de Simulink dando un mal aspecto al resultado final.

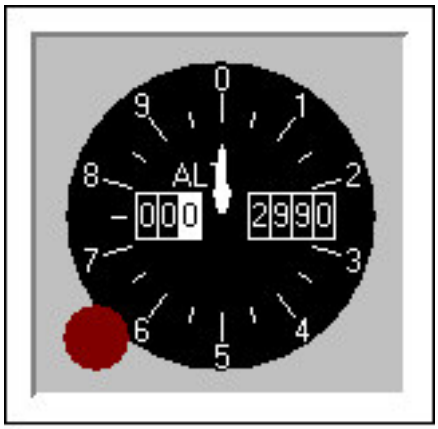
El bloque Active X tiene una o varias propiedades dependiendo de las entradas que posea (input property).En estas propiedades guarda el valor que luego va a pasar al dial que se encuentra en un subsistema aparte (Handle location) y así este dial tendrá el valor a mostrar.



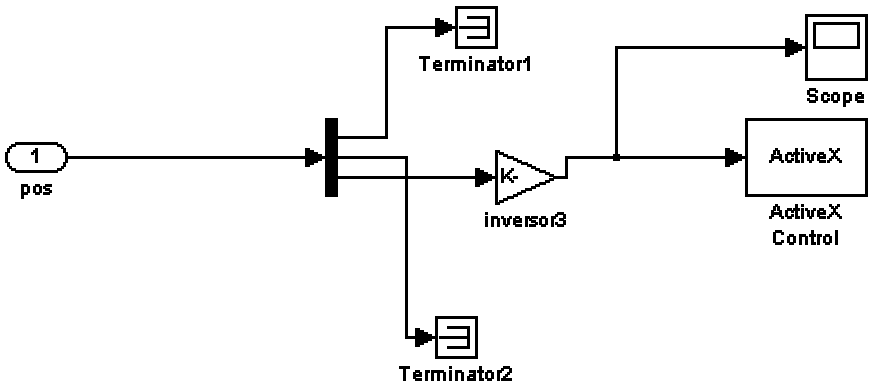
A continuación mostramos los diales utilizados y sus correspondientes bloques de Simulink.

6.2.5.1 Altímetro

En este caso tenemos que coger la coordenada de la altura. En el bloque de SIMULINK tenemos la posición y cogemos la tercera coordenada que es con nuestro cambio de ejes la que nos da la posición en vertical. Luego la pasamos al bloque Active X. Las unidades en las que se mide son metros.

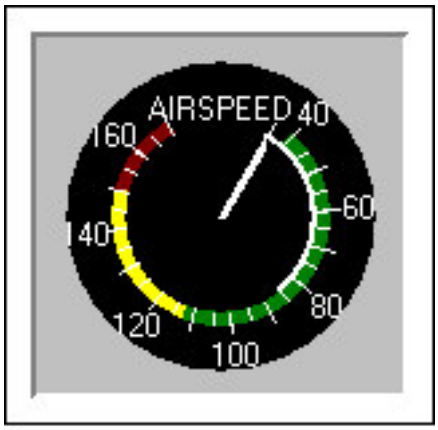


Altimeter

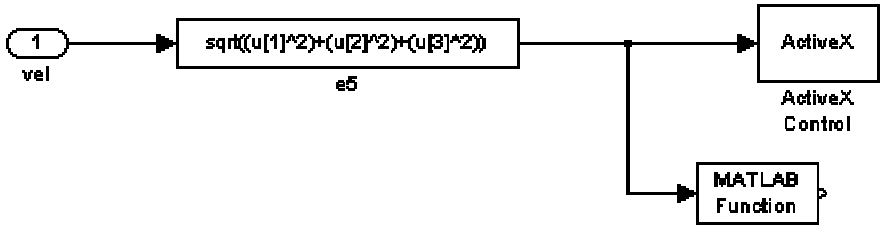


6.2.5.2 Velocímetro

En el velocímetro lo que nos interesa saber es la velocidad en los tres ejes y así sacar la velocidad total. Por medio de la formula SQRT($x^2+y^2+z^2$) la obtenemos y la pasamos al bloque ActiveX. Las unidades en las que se representa la velocidad son m/s.

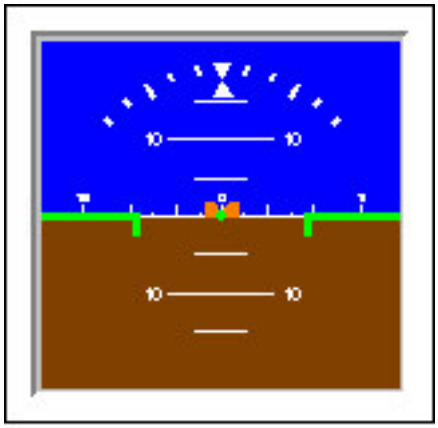


AirSpeed

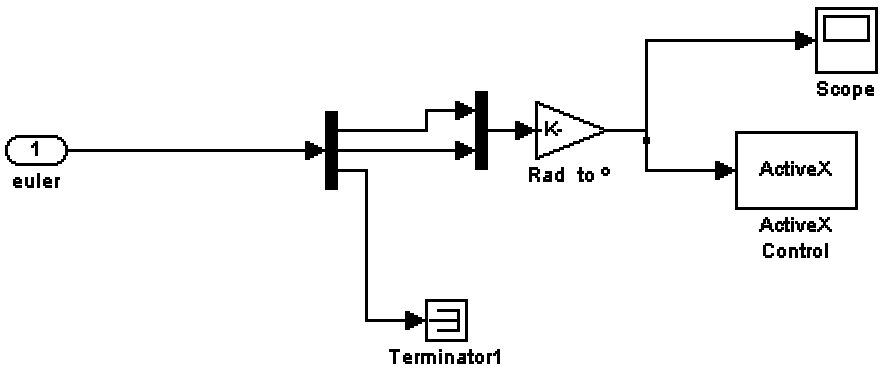


6.2.5.3 Horizonte Artificial

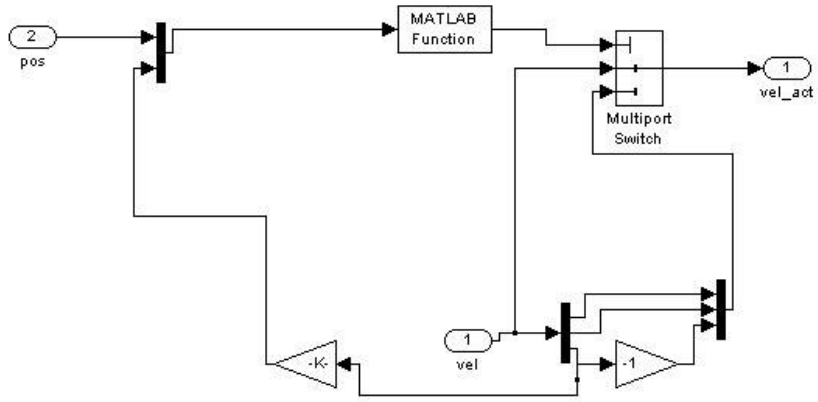
En el horizonte artificial necesitamos dos de las tres ángulos de euler, en este caso phi y theta, ya que estos nos dan el ángulo de la inclinación longitudinal del helicóptero (Pitch) y el ángulo de la inclinación lateral del helicóptero (Roll). Ya solo queda pasar el ángulo de radianes a grados y pasarlo al bloque ActiveX.



Artificial Horizon



6.2.6 COLISIONES



Las colisiones surgen para evitar que el helicóptero atraviese el mundo.

La idea sería por tanto encontrar un mapa con las alturas del terreno para poder saber cual es el valor que no debe pasar.

Para comparar el valor de la altura del terreno en cada punto con un valor del helicóptero, se podría utilizar el punto medio del mismo, pero en este caso si alguna parte del exterior(hélices, cola) chocasen con el terreno no se detectaría hasta que no lo hiciera el punto medio.

Por tanto decidimos meter el helicóptero dentro de un rectángulo, y así hacer que lo que choque sea el rectángulo en lugar del propio helicóptero. Para ello se cogen los ocho puntos que formarían las esquinas del rectángulo, y son las alturas de dichos puntos las que se comparan con las alturas del terreno.

El envolver el helicóptero en un rectángulo se hace de una manera bastante sencilla, ya que tenemos la posición del mismo, con lo cual bastaría con sumarle y restarle unos valores para formar los ocho puntos.

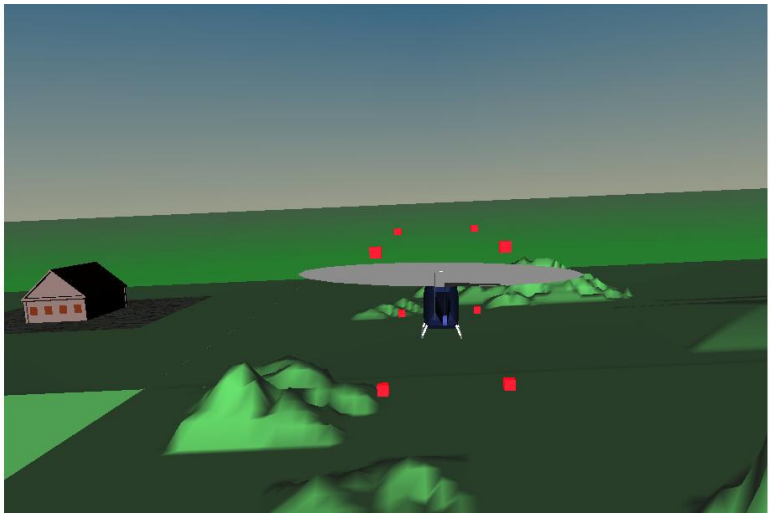
El problema se plantea a la hora de mover el helicóptero, ya que los ocho puntos se deberían mover en la misma dirección y con el mismo ángulo de giro. La función que calcula los ocho puntos sería *Calcula_puntos,* la cual llama a la función *calcula_Variaciones*.

La función *calcula_Variaciones* dado un ángulo de giro que pueda ser el pitch, el roll o el yaw, se calcula por trigonometría la variación de sus dos coordenadas que se ven afectadas por dicho giro. Debido a que cada giro se realiza en uno de los tres planos:

- Roll afecta a las coordenadas x e y
- Pitch afecta a las coordenadas z e y
- Yaw afecta a las coordenadas x e z

Para calcular los ocho puntos del rectángulo, hay que tener en cuenta que la posición que recibimos del helicóptero, viene en coordenadas absolutas, respecto de un origen en el mundo, mientras que esa variación que nos calculamos viene referida respecto del ángulo de giro del helicóptero, esto es respecto del eje central del helicóptero, por tanto necesitamos calcularnos las coordenadas del centro.

Por tanto para calcular esos puntos se cogen las coordenadas de la posición del helicóptero, se suman los valores para llegar a los extremos, y se le suman esas variaciones de giro. Aunque posteriormente los ocho puntos no sean visibles el rectángulo que envuelve al helicóptero sería el siguiente:



Otro de los objetos necesarios para calcular las colisiones, es el mapa de colisiones, que es un vector con todas las alturas del terreno. Para sacar el mapa de colisiones es necesario utilizar el código con el que el editor gráfico guarda sus archivos. Éste código se puede ver abriendo el archivo con el wordpad, y descubrimos que cada nodo viene representado por su nombre y unos determinados atributos.

En el caso del nodo terreno, hay un atributo llamado *height* que tiene todas las alturas de todos los puntos del terreno. Como hemos descrito anteriormente tenemos 9 terrenos, mientras que le mapa de colisiones es el vector de sólo un terreno, por ello al cambiar de terreno se calcula su nuevo mapa.

Con la variable *actual*, sabemos el terreno en el que se encuentra el helicóptero, con lo que al ir a mirar al código del editor, tenemos que buscar el nodo correspondiente, es decir, en caso de que *actual* sea 5 buscaremos donde ponga terrano5.

Una vez localizado el punto donde empieza las alturas se pasa a una variable array de matlab. Esta variable la guardamos como un vector de una fila y 10000 columnas.

Por último, se procede a comparar las alturas mediante la función *Altura_actual*.

Una vez que tenemos los ocho puntos del rectángulo y el mapa de colisiones, se compara cada uno de los puntos con su valor en el mapa de colisiones.

Para calcular el valor del punto en el mapa, se le aplica el modulo 99 a su coordenada x, y a su coordenada z, para que cuadre siempre dentro del terreno aunque éste se haya movido debido al mundo infinito. Posteriormente se aplica la fórmula:

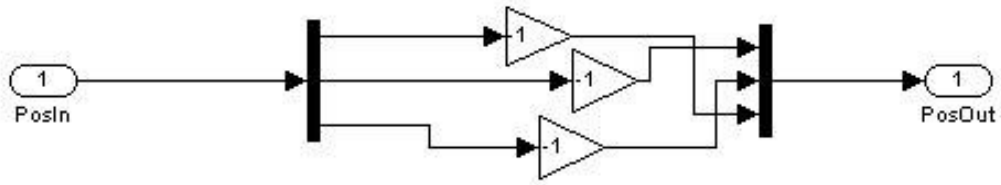
mapa_Colisiones(z*100+x)

Con esta fórmula hacemos que se coja la altura adecuada para cada punto, ya que están colocados en el mapa las 10000 posiciones seguidas empezando en la esquina superior izquierda y terminando en la esquina inferior derecha.

El valor que se compara con la altura del mapa colisiones, no es exclusivamente el de la posición del punto, sino que además hay que añadirle la velocidad del helicóptero, para que se calcule la posición siguiente del helicóptero, y en caso de colisión evitar llegar a esa posición.

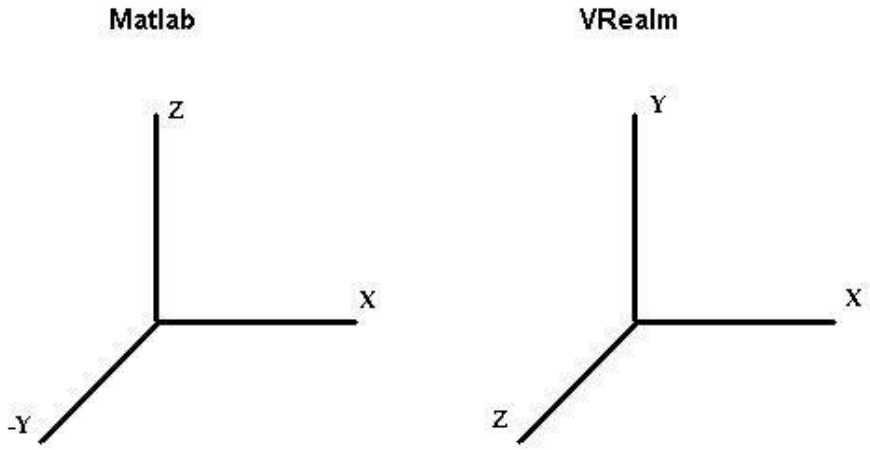
En cuanto al modelo de simulink, se utilizan estas funciones de matlab descritas anteriormente, y en caso de que no se produzca colisión se mantiene la misma velocidad que lleva el modelo, mientras que si se produce una colisión, se invierte la velocidad en el eje z.

6.2.7 EJES

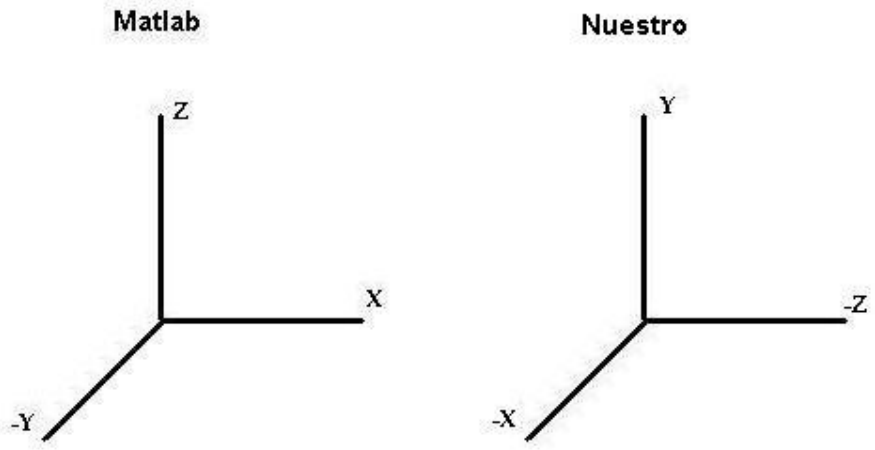


Lo primero para entender el cambio de ejes que se debe realizar a la posición del helicóptero antes de trabajar con ella, para después mandársela al visor del mundo virtual, es que los ejes utilizados por Matlab y los ejes utilizados en VRML (Virtual Reality Modeling Language) son diferentes. Por consiguiente los ejes de coordenadas del editor VRealm también lo son.

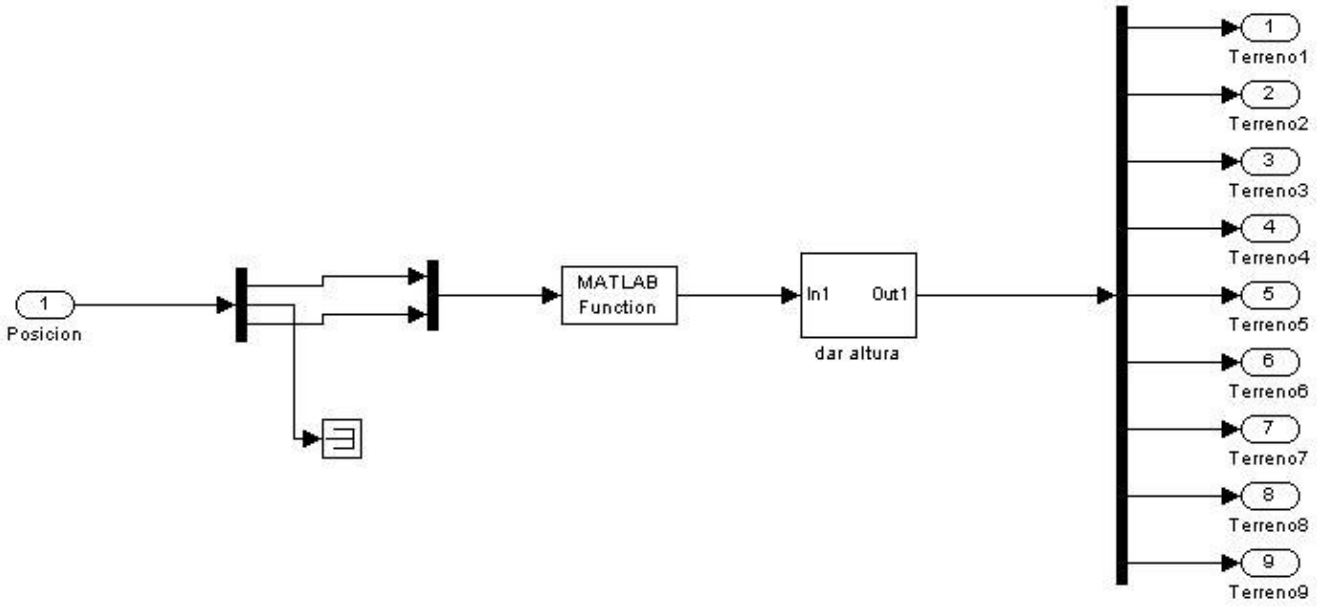
Vemos aquí esa correspondencia.



Como bien se puede observar, esta correspondencia no equivale exactamente a nuestra transformación, esto se debe a que el helicóptero en su posición inicial esta mirando hacia las –Z, por eso cuando en Matlab aumentan, nosotros debemos disminuirlas para que el helicóptero avance. Esto añadido a que preferíamos trabajar con las equis y las zetas intercambiadas, hizo que nos decidiéramos por esta otra transformación.



6.2.8 MUNDO INFINITO



Corresponde al módulo Transformación Terreno del modelo en simulink.

Aquí explicaremos cómo hemos conseguido que el mundo sea infinito. Partíamos con unos problemas, que eran estos:

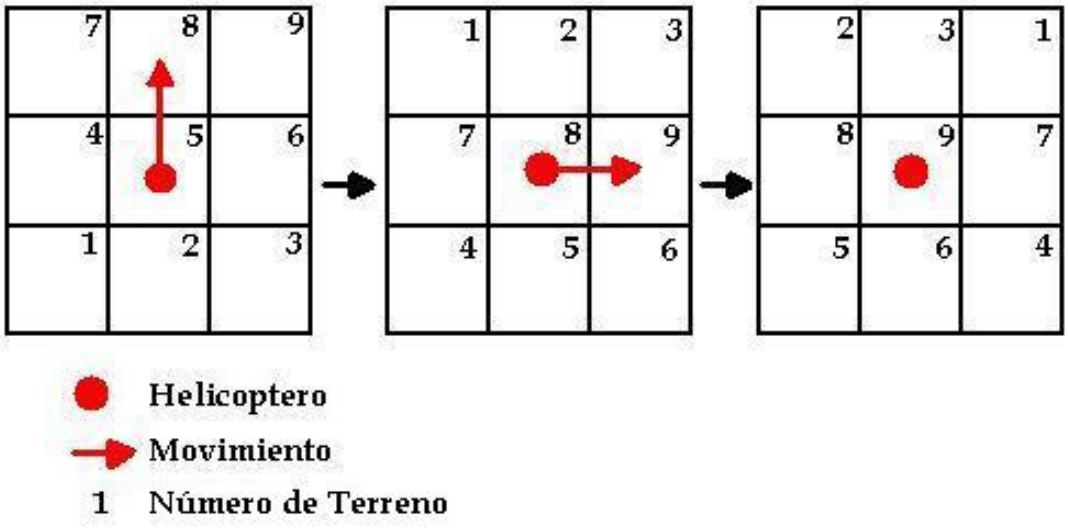
1. En el editor teníamos que tener los terrenos ya hechos, no se pueden crear en ejecución.
2. Cada terreno solo puede ser de 100 x 100 nodos.

El segundo punto implica que no podíamos hacer un terreno gigantesco, que fuera lo bastante grande como para que no hiciera falta más. Eso hace que tuviéramos que plantearnos hacer varios terrenos para un mismo mundo. El primero, implica que deban estar hechos, impidiendo que los hagas cuando los necesites.

Con estos problemas, decidimos usar una solución bastante sencilla, pero no muy habitual. Consistía en mover los terrenos según nos fuéramos moviendo nosotros con el

helicóptero. Claro que no se puede hacer que se muevan exactamente contigo, porque entonces no daría la impresión de movimiento.

Por eso decidimos tener 9 Terrenos de mismo tamaño, formando un mundo cuadrado, y que el helicóptero se encontrara siempre en el terreno de en medio, pudiendo moverse con libertad, pero que al cambiar de terreno, se recolocaran de forma que siguieras en el de en medio. Pero siempre guardando el orden, es decir que si volvieras sobre tus pasos, te encontrases con lo mismo. De esta forma:



Para realizar esto, realizamos la función de Matlab, colocamapa(Posicion), Posicion es la posición actual del helicóptero. Esta función usa las variables globales Terreno y actual. Terreno es un vector con las coordenadas [x , z] de cada uno de los 9 terrenos y actual es el número del Terreno en el que se encontraba en el instante anterior el helicóptero.

Con esto ya solo nos hace falta ver si sigue en el mismo Terreno, en ese caso no haríamos nada, devolveríamos Terreno y actual se quedaría como está. En caso contrario miramos a que terreno ha ido el helicóptero, lo ponemos como actual, y variamos las x y z de los terrenos correspondientes.

Para variar las coordenadas [x, z], consideramos que el nuevo actual es el centro de los otros 8 terrenos. Teniendo en cuanta esto, y que los terrenos tienen solo 100 nodos, debemos asegurarnos de que todas las coordenadas de un terreno están a menos de 100 nodos de las coordenadas del central. También sabemos que las coordenadas deben ser múltiplos de 99, puesto que para que no haya ninguna fisura entre los terrenos, el último nodo de un terreno debe coincidir con el primero del siguiente terreno.

Para entenderlo mejor pongamos un ejemplo. Supongamos que el Terreno1, es el central y que su coordenada X es igual a 0. Todos los terrenos que en la X estén al lado suya serán los que tengan de coordenadas X∈ {-99, 0, 99}.

Supongamos que el Terreno5 tiene de coordenada X el valor 188, como se pasa por encima, le ponemos como nueva coordenada X el valor de la X del Terreno1 menos 99, por lo que ahora su nuevo valor sería –99, y ya pertenecería al rango aceptable.

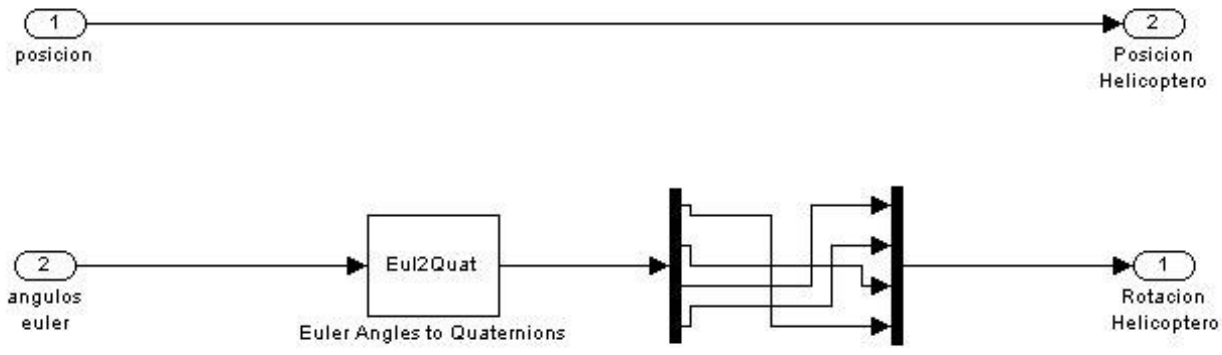
Vemos las líneas de código más importantes, que nos permiten profundizar en este pensamiento, Tact es el terreno actual:

```
%Ya tenemos donde estamos, ahora colocar el resto de terrenos
   %con respecto a este.
   for j=1:9
      Tx = Terreno((j*2)-1);
      Ty = Terreno((j*2));
      aux1 = Tact(1)-Tx;
      aux2 = Tact(2)-Ty;
      if aux1>99
         Terreno((j*2)-1) = Tact(1)+ 99;
```

```
    end
    if aux1<(-99)
       Terreno((j*2)-1) = Tact(1)- 99;
    end
    if aux2>99
       Terreno((j*2)) = Tact(2)+ 99;
    end
    if aux2<(-99)
       Terreno((j*2)) = Tact(2)- 99;
    end
end
```

Esto es básicamente lo que hace la función colocamapa. Una vez que tenemos las coordenadas [x, z], ya solo queda añadir la coordenada 'y', siempre a 0, y mandarle todas las posiciones al nodo Traslation de cada uno de los terrenos del mundo virtual. De esto se encarga el bloque "dar altura".

6.2.9 TRANSFORMACIÓN HELICÓPTERO



En este apartado hablaremos de la transformación que se debe hacer de la posición y ángulos del helicóptero antes de mandárselos al Mundo virtual.

Como podemos observar gracias al cambio de ejes explicado anteriormente la posición del helicóptero ya nos llega perfectamente. Sin embargo los ángulos hay que tratarlos. Aquí nos llegan los ángulos de euler (roll, pitch y yaw) y Vrealm usa cuaterniones.

Lo primero es explicar que son los cuaterniones. Un vector cuaternión representa una rotación con respecto a un vector unidad (μ_x, μ_y, μ_z) de un ángulo θ. Un cuaternión por el mismo tiene magnitud, y puede ser escrito con el siguiente formato.

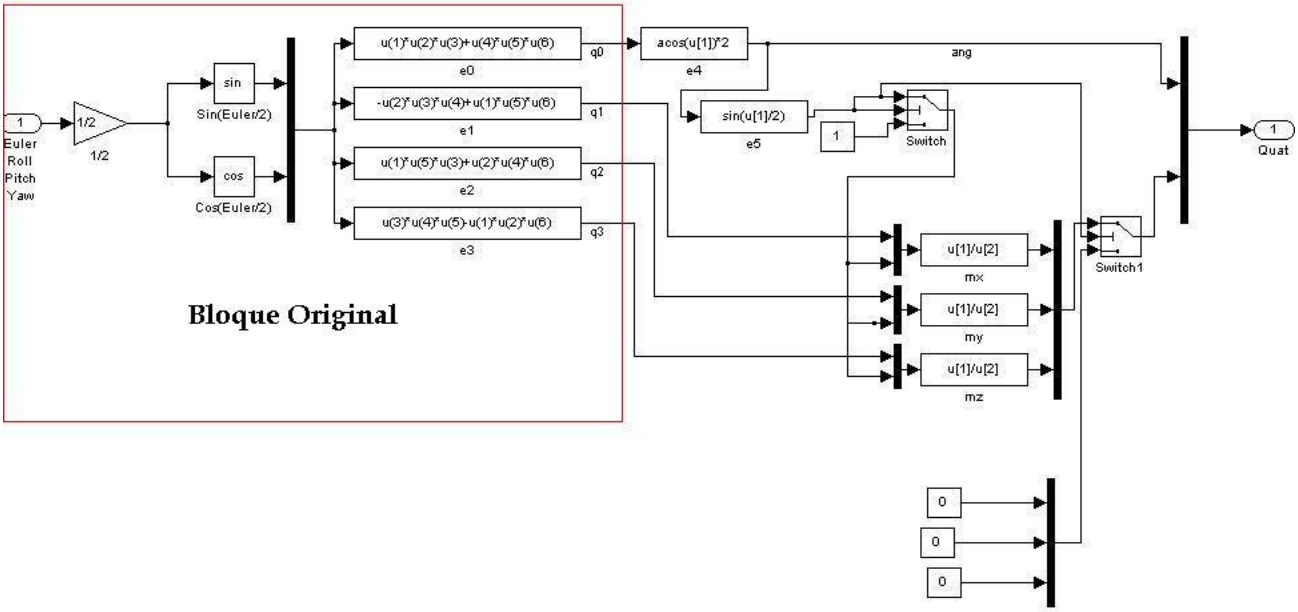
$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)\mu_x \\ \sin(\theta/2)\mu_y \\ \sin(\theta/2)\mu_z \end{bmatrix}$$

Otra posible representación es esta.

$$q = q_0 + iq_1 + jq_2 + kq_3$$

El vector formado por (q_0, q_1, q_2, q_3) es lo que teóricamente devuelve la función de Simulink "Euler Angles to Quaternions", mientras que VRealm necesita que le llegue el vector formado por $(\mu_x, \mu_y, \mu_z, \theta)$

Para realizar esto hicimos una pequeña variación en el bloque antes mencionado para poder obtener el vector deseado.



Simulink usa el vector (q_0, q_1, q_2, q_3) que en principio parece menos lógico, puesto que son solo valores en función de los importantes, pero es así para poder evitarse singularidades, divisiones por cero para ser exactos.

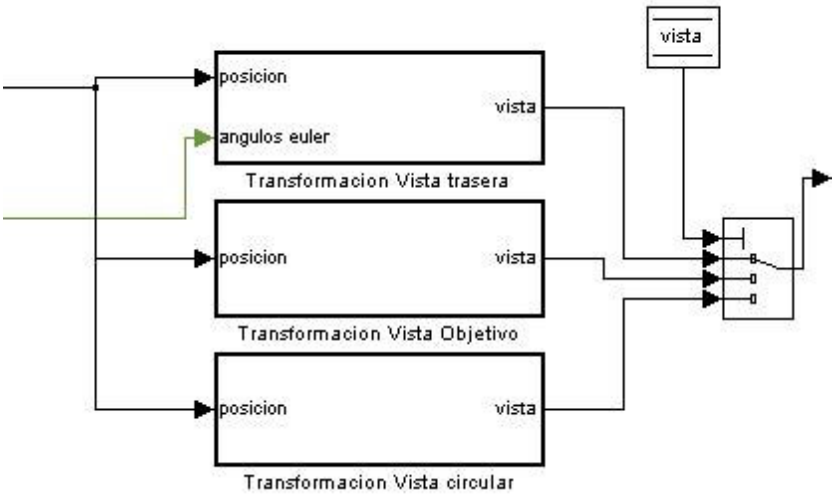
Las fórmulas que usamos para hallar los que nosotros necesitamos son estas:

$$\theta = \text{acos}(q_0) \times 2$$
$$\mu_x = q_1 / \sin(\theta/2)$$
$$\mu_y = q_2 / \sin(\theta/2)$$
$$\mu_z = q_3 / \sin(\theta/2)$$

Aquí podemos ver que en el estado inicial, cuando la rotación es de 0 grados, sin(0) vale cero y se produce un error de división por cero. Por esta razón el módulo es mucho más lioso de lo esperado. Porque nos hemos visto obligados a mirar esos casos en los que en ángulo es cero.

La solución que hemos puesto para resolver esa singularidad, es devolver el vector (0, 0, 0, θ), en caso de que se fuera a producir. Los dos switches de la imagen, son los que implementan la especie de bloque if necesario para resolverla.

6.2.10 VISTAS



En este apartado hablaremos sobre las 3 cámaras que tiene nuestro simulador para ver la acción que se desarrolla en el mundo virtual. Todas ellas tienen como centro y objetivo de la imagen, nuestro helicóptero.

Tenemos la vista trasera, que es la vista tradicional de todo simulador de vuelo en el mercado. La vista del objetivo, que mira hacia el siguiente objetivo y la vista móvil o circular, que te permite manejarla a tu antojo.

Antes de tratar otros temas deberíamos comentar como son las cámaras en el VRealm Builder. En el editor se conocen como puntos de vista o Viewpoints. Los nodos Viewpoint definen una localización especifica en el sistema de coordenadas local desde la que el usuario puede ver la escena. Puede haber tantos Viewpoints en la escena como se quiera, pero solo uno puede estar activo al mismo tiempo.

Haremos una breve descripción de los principales parámetros que tienen estos nodos:

- ***fieldOfView*:** Es más o menos el campo de visión que abarca. Los valores que admito, son números reales de 0 a pi, que indican el ángulo de visión de la cámara. Lo que la cámara hace, cuando le restringes mucho el ángulo es avanzarse, y cuando se lo abres mucho, alejarse. El valor que tiene por defecto, y que parece bastante adecuado es 0.7853, que equivale a un ángulo de 45º.
- ***orientation***: Es la orientación de la cámara dada en cuaterniones (μ_x, μ_y, μ_z, θ), los 3 primeros valores que necesita, forman la x,y,z del eje sobre el que la cámara va a girar θ grados.
- ***position**:* Es la localización especifica de la cámara u objetivo en el sistema de coordenadas local. Dada por un vector de coordenadas (x, y, z).
- ***description***: Aquí se puede poner una descripción básica de este objetivo. "Vista trasera" por ejemplo.
- ***set_bind*:** Es un parámetro booleano, que si esta a true, pone esa cámara como activa. Por nuestra experiencia, si hay varias a true, la última en tomar ese valor será la activa.

Otro tema a tratar es la selección de la vista, en el editor de mundos virtuales (VRealm) hemos tenido que añadir los nodos Viewpoints, cada uno de estos es una cámara. En el modo de uso con el joystick virtual, se cambia muy sencillo de una vista a otra. Puesto que la misma ventana donde se ve el mundo virtual tiene unos botones para cambiar las vistas.

Con esos dos botones podemos ir moviéndonos por las vistas sin problemas. El problema llega en el modo de uso con teclado, en donde, como hemos explicado anteriormente, debe estar siempre activa y por encima de las demás ventanas, la ventana que nos va a capturar las teclas pulsadas. Esto imposibilita que podamos pulsar con el ratón en esos dos botones para manejar las vistas, a menos claro , que deshabilitemos por un momento la entrada por teclado.

Para solucionar este problema no nos quedó otra, que usar una sola vista, con la que empezara por defecto el visor, en nuestro caso el Viewpoint2.

Pero queríamos seguir manteniendo todas las vistas, y la solución no fue tan difícil, bastaba con según que vista eligiéramos por teclado, mandarle a un mismo Viewpoint unas posiciones diferentes. Entonces, en vez de, como en el modelo con joystick, tener 3 cámaras y elegir entre ellas, en el de teclado, tenemos una cámara y elegimos la posición y orientación que mandarle. Para realizar esto, usamos la variable global “vista”, como entrada a un switch, esto es lo que podemos observar en la captura de la página 63.

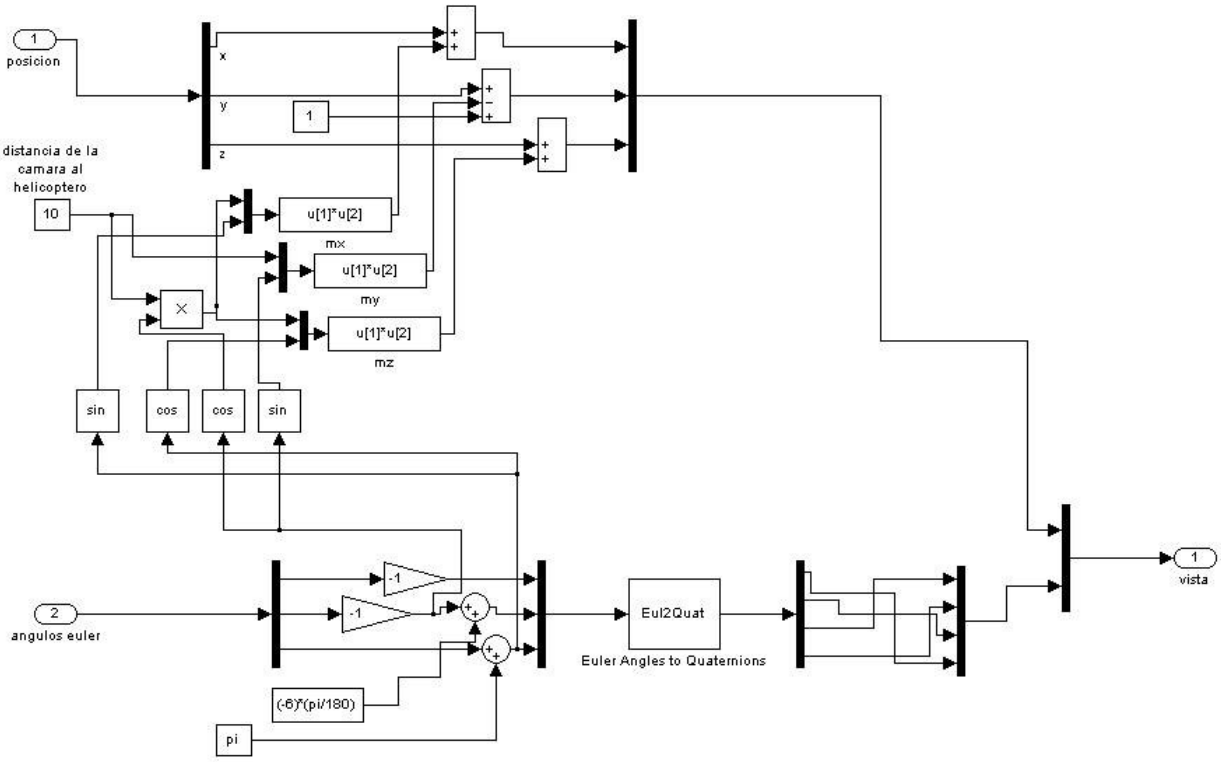
Ahora que hablamos lo de la posición y orientación, decir que son los únicos parámetros que trabajamos de las cámaras, hay otros posibles como su ángulo de apertura y demás, pero preferimos dejarlos como por defecto, puesto que no aportaba gran cosa.

La descripción básica de las ecuaciones utilizadas para hallar estos dos parámetros, la haremos sobretodo para la primera vista, la trasera. Puesto que más o menos el resto van en función de esta.

En el modelo con entrada por teclado estas son las teclas para cambiar de una vista a otra:

- ***'7'*** : Vista trasera
- ***'8'*** : Vista Objetivo
- ***'9'*** : Vista Circular

6.2.10.1 Vista trasera



Como entradas recibimos la posición y orientación del helicóptero, y a partir de ahí debemos obtener la posición y orientación de la cámara. La distancia de la cámara al helicóptero es de 10 metros.

Empezaremos explicando, lo que se refiere a la **orientación**. Como queremos que la cámara mire hacia el mismo sitio que el helicóptero, esta debe tener una orientación casi idéntica.

Por esto, el tratamiento que se le aplica a los ángulos de euler es similar al que se le aplica a dichos ángulos en el modulo “Transformación Helicóptero”.

Como la posición inicial de los Viewpoints en el VRealm son justo al revés que la posición inicial del helicóptero, debemos sumarle ‘π’ al yaw que llega de entrada. Como es

lógico esta inversión del punto de vista, crea un efecto espejo, que subsanamos cambiando de signo el roll y el pitch.

Además, no queríamos que se viera justo desde detrás del helicóptero, sino que era más visual desde un poco más arriba. Por esto le restamos al pitch, 8º. De esta forma la cámara esta un poco más inclinada hacia abajo que el helicóptero. Que, con lo que veremos ahora de cómo hallamos la posición, explica que el objetivo, se encuentre más elevado (hacia arriba normalmente, pero podría ser hacia abajo en caso de looping).

Después de estos cambios ya solo se les hace falta, a los ángulos de euler, el mismo tratamiento que en el módulo "Transformación Helicóptero", pasándolos a cuaterniones que es lo que usa el nodo Viewpoint del VRealm.

En cuanto a la **posición**, ya no solo depende de la posición del helicóptero, al igual que la orientación solo dependía de la orientación del helicóptero, sino que la posición también depende de la orientación de la cámara.

Básicamente es un problema de trigonometría en 3 dimensiones. La cámara y el helicóptero forman un triángulo, cuyo vértice inferior, es la posición del helicóptero, la hipotenusa es la distancia entre la cámara y el helicóptero (10 metros como hemos comentado antes) y los ángulos restantes se puede sacar del roll(ϕ), pitch(θ) y yaw(ψ) de la cámara, puesto que queremos que esta, esté centrada en el helicóptero o vértice inferior.

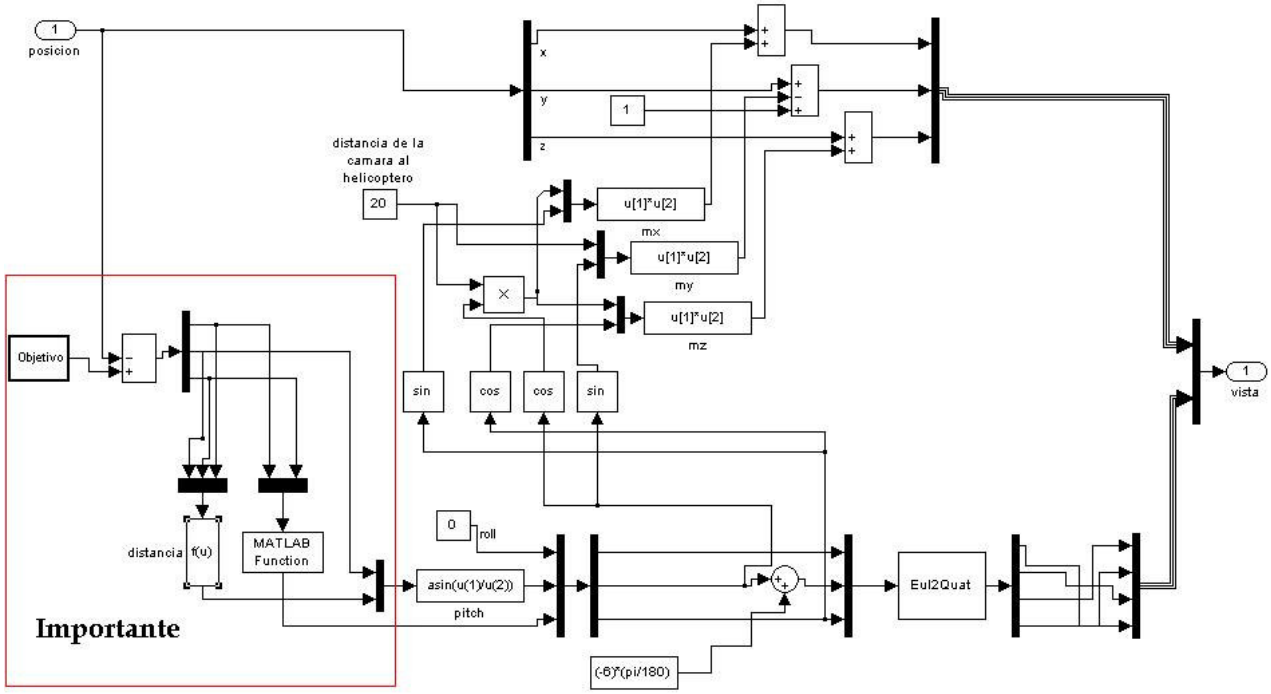
Como se ve fácilmente, para la posición de la cámara no influye para nada el roll, del helicóptero, solo el pitch y el yaw. Y la altura, o componente "y" de la cámara solo depende del pitch, mientras que para la "x" y la "z", influyen tanto el pitch como el yaw.

Quedando de esta forma las ecuaciones resolutorias:

$$x = \sin(\Psi) \times \cos(\theta) \times 10$$
$$y = \sin(\theta) \times 10$$
$$z = \cos(\Psi) \times \cos(\theta) \times 10$$

Aparte de esto ya solo queda sumarle 1 a la “y”, para que la cámara este incluso más arriba.

6.2.10.2 Vista Objetivo:



Esta vista, es en la que se ve, además de al helicóptero, al siguiente objetivo. Para que se pueda ver mejor el objetivo, y no se oculte detrás del helicóptero, debimos separar la cámara, en vez de a 10 metros del helicóptero, a 20. De esta forma, las fórmulas resolutorias para hallar la posición después de tener ya, los ángulos de euler son diferentes. Estas son:

$$x = \sin(\Psi) \times \cos(\theta) \times 20$$
$$y = \sin(\theta) \times 20$$
$$z = \cos(\Psi) \times \cos(\theta) \times 20$$

El resto de cálculos para la posición de la cámara son exactamente idénticos a los realizados en el módulo “Transformación Vista trasera”.

Estas fórmulas son para una vez que se tienen los ángulos de euler, que en la vista trasera, eran una de las entradas, y correspondían al roll, pitch y yaw del helicóptero. Mientras

que en esta vista, ya no importa como este el helicóptero, sino tan solo donde estén, tanto el helicóptero como el siguiente objetivo.

La posición del helicóptero es la entrada del módulo, y la del objetivo, la hemos guardado anteriormente, concretamente, en el módulo de rumbos, es donde se lee desde fichero, cuales son esos objetivos, y es así donde se guarda en esta variable de Simulink.

Nos hallamos otra vez ante un problema de trigonometría en 3 dimensiones. Lo primero hallamos el vector que va desde el objetivo al helicóptero, restando simplemente a las coordenadas x, y, z del objetivo, las del helicóptero. Tenemos:

$p = \{x, y, z\}$

Una vez con esto, necesitamos saber el pitch(θ) y el yaw(ψ) necesarios de la cámara, para que esté colocada en ese vector. Otra vez el roll no nos importa para nada, puesto que no influye en el vector, sino en la rotación de ese vector sobre si mismo. Por esta razón ponemos el roll a 0, puesto que queremos ver el objetivo paralelos al suelo.

Primero nos ocuparemos del pitch, que es algo más sencillo, para esto, básicamente, con saber la distancia a la que esta y la altura nos bastaba. Quedando las fórmulas necesarias de esta forma:

$$d = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = \sin^{-1}(y / d)$$

En cuanto al yaw ya era más complicado, porque dependía del signo de la x y de la z, dependiendo de cual cogieras de las dos variables, solo podías cubrir 180º de la circunferencia.

Al final optamos por hacerla en una función nuestra escrita en Matlab, para verla con más detalle, es la función "yaw". Las fórmulas básicas son estas:

$$d = \sqrt{x^2 + z^2}$$

$$\Psi = \cos^{-1}(z / d)$$

Pero una vez hecho esto, debimos modificarlo, para que tuviera en cuenta todos los ángulos posibles, y cubriera la circunferencia completa. Según cuales fueran los valores de 'x' y 'z' debíamos modificar el ángulo de cierta forma. El ángulo en si, ya lo tenemos una vez hechas las fórmulas, ya solo queda ponerlo en su posición correcta, cambiándolo de signo o sumándole y restándole π radianes. Lo mejor para entenderlo perfectamente es mostrar el código de la modificación:

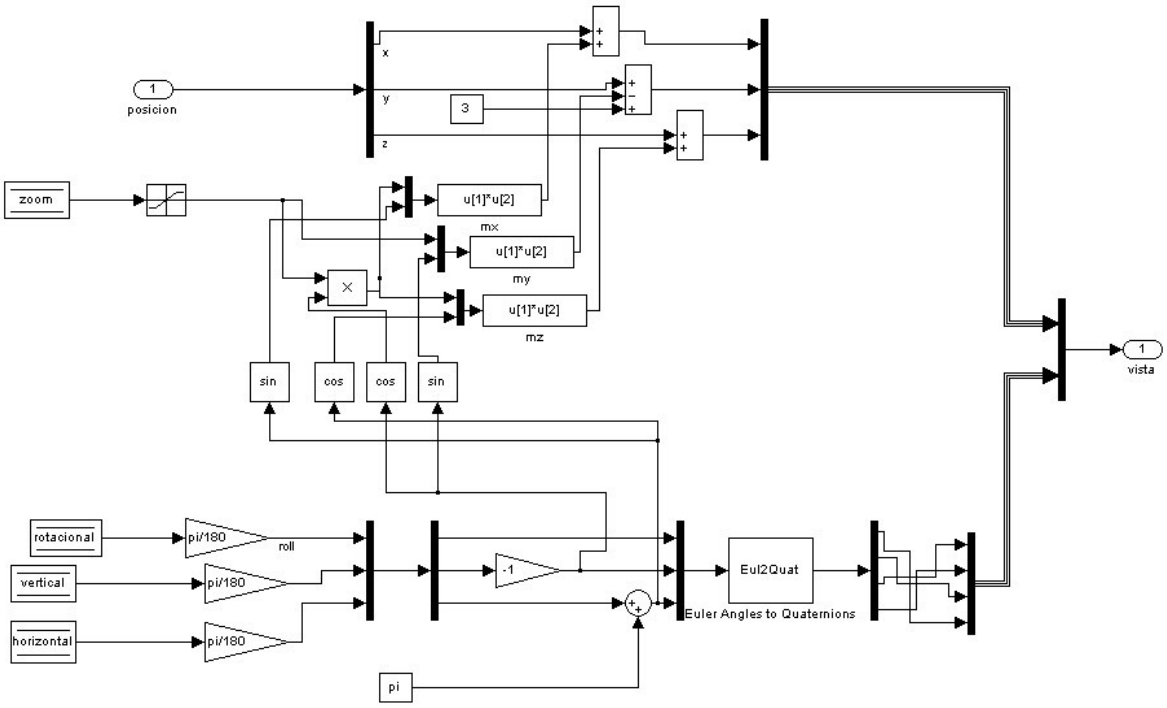
```
if (z>0) && (x>0)
   ang = pi + ang;
end;
if (z>0) && (x<0)
   ang = pi - ang;
end;
if (z<0) && (x>0)
   ang =  - ang;
end;
if (z<0) && (x<0)
   ang = ang;
end;
```

Como los cálculos de los ángulos de euler los hemos hecho nosotros en vez de que nos llegaran como entrada, al contrario que en la vista trasera, ya nos es necesario sumarle π al yaw, y por lo tanto tampoco cambiar de signo al pitch y al roll.

Le restamos al igual que en la vista trasera, 6º al pitch, para ver la imagen un poco más inclinada.

Pasamos los ángulos a cuaterniones y llegamos a la vista que vemos aquí abajo.

6.2.10.3 Vista Circular



Como vemos, el esquema es exactamente el mismo que en la vista trasera. Solo que la única entrada es la posición del helicóptero. Lo demás son todo las variables que llegan desde teclado, como en la imagen, o con sliders en el modelo con joystick virtual. Esta variables son el roll, el pitch, el yaw y la distancia de la cámara al helicóptero. Básicamente, esta cámara te permite moverte por toda la circunferencia, con el helicóptero como centro, y el zoom como radio.

El zoom tiene un límite de 240 como máximo y de 8 como mínimo. Puesto que la posición de la cámara depende del zoom, estas son sus ecuaciones:

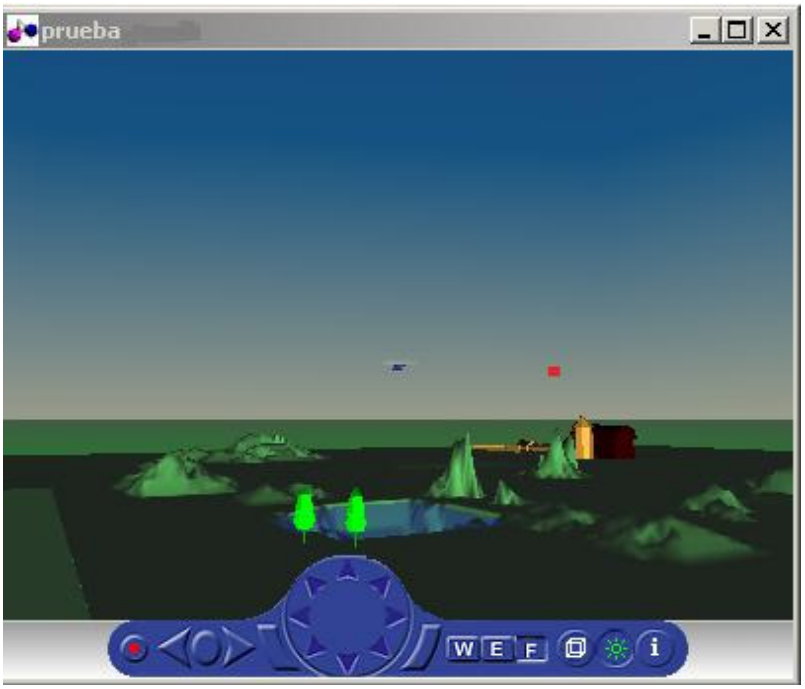
$$x = \sin(\Psi) \times \cos(\theta) \times zoom$$
$$y = \sin(\theta) \times zoom$$
$$z = \cos(\Psi) \times \cos(\theta) \times zoom$$

En esta ocasión si que hemos sumado π al yaw y cambiado de signo el pitch, tal como se hacía en el módulo "Transformación Vista trasera".

En el modelo con entrada por teclado estas son las teclas para manejar esta cámara:

- ***'J' y 'L'***: para moverte horizontalmente por la circunferencia, o lo que es igual, cambiar el yaw.
- ***'I' y 'K'*** : para moverte verticalmente, o cambiar el pitch.
- ***'Y' y 'H'***: para girar sobre la cámara sobre si misma, o mover el roll.
- ***'U' y 'O'*** : para ampliar o disminuir el zoom respectivamente.

Estas son unas muestras de lo que se puede hacer con esta cámara.




prueba
W
E
F

7 CONCLUSIONES Y POSIBLES MEJORAS

A lo largo de este trabajo, se ha intentado en todo momento llevar al límite las posibilidades gráficas de Matlab y Simulink. Aprendiendo a manejar las herramientas Matlab y Simulink, así como unas nociones sobre helicópteros y aerodinámica. Realizando un trabajo de una magnitud, en lo que se refiere a control y visualización, nada común en estos entornos. Con esta idea se ha construido un modelo en Simulink que ofrece muchas posibilidades en la continuación de este trabajo.

Según se ha ido avanzando hemos ido viendo mayores posibilidades de mejora. Tanto en el aspecto gráfico como físico del modelo.

En el apartado estético del simulador, se podría trabajar en la creación de diales propios para Simulink. Por otro lado, la inclusión de sonidos y animaciones(como explosiones), podría dar mayor realismo a la simulación. Se podría aumentar el número de mundos, así como alguna opción para cambiar el tipo de helicóptero.

En el aspecto físico del modelo, aspectos como el tiempo o el viento podrían ser parametrizables, al igual que se podría hacer con los posibles tipos de helicóptero. La mejora del control manual podría ser interesante, así como la perfección del piloto automático.

Aunque sobretodo, este trabajo puede ayudar en la creación de maquetas como las que aquí se simulan, y perfeccionar en este simulador sus métodos de control, antes de intentar aplicarlos en la realidad. Por esto cuanto más real llegue a ser, más útil podrá ser.

8 AGRADECIMIENTOS

Agradecemos su ayuda, tiempo y disponibilidad a:

- Fernando Saénz Pérez, Departamento de sistemas informáticos y programación. UCM: por habernos guiado a lo largo de todo el año y habernos dado la oportunidad de realizar este proyecto. Dándonos siempre las mayores facilidades y toda su disponibilidad.
- Guillermo Martínez Sánchez, Departamento de arquitectura y automática UCM: por habernos dejado el modelo fruto de un proyecto anterior suyo, y habernos ayudado en lo posible.

9 AUTORIZACIÓN

Por la presente autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto como la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Madrid, a 27 de Junio de 2005.

Mario Chueca Burgueño,

Javier López Carreras,

Iván Rebollo Martínez

10 PALABRAS CLAVE

- Helicóptero
- Simulador
- Matlab
- Simulink
- Juego
- Modelado
- Vuelo
- Mario
- Iván
- Javier

11 BIBLIOGRAFÍA

- V·Realm™Builder
  User's Guide and Reference
- Ayuda de matlab:
  1. Virtual Reality Toolbox for use with MATLAB and SIMULINK
  2. Aerospace Blockset for use with SIMULINK
  3. Dials & Gauges Blockset for use with SIMULINK.
- http://3dplants.0catch.com/download.html
- Nelson. R.C. Flight Stability and Automatic Control 2nd Edition McGrawHill 1998.
- http://www.3dcafe.com/
- MODELACIÓN DEL SISTEMA NO LINEAL DE UN HELICÓPTERO – proyecto del departamento de arquitectura de computadores y automática de la UCM.
- http://www.mathworks.com/matlabcentral/
- T. John Koo, Yi Ma, and S.Shankar Sastry Nonlinear Control of a Helicopter Based Unmaned Aerial Vehicle Model.
- Flying Model Simulator 2.0